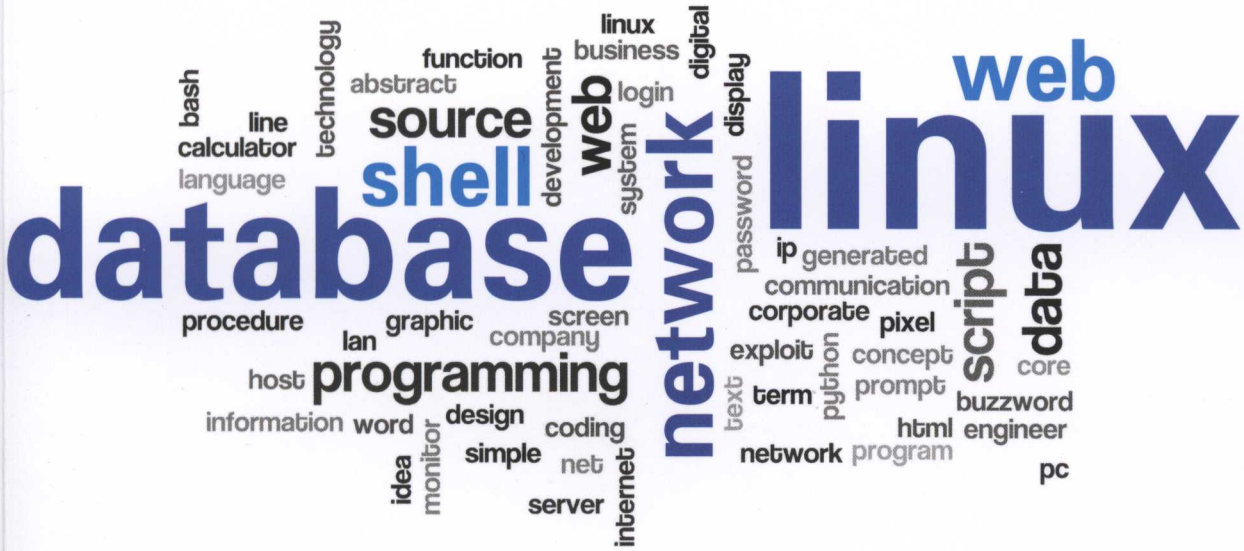
版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。

每时每课 给你新机会

课工场 kgc.cn

云计算工程师系列

北京课工场教育科技有限公司 出品



◉ 视频课程 ◉ 案例素材 ◉ 交流社区 ◉ QQ讨论组

数据库应用

主 编 肖 睿 于继武
副主编 杨高岭 刘玉强 李 震

中国水利水电出版社
www.waterpub.com.cn

课工场介绍

课工场是专注互联网教育的生态平台，汇聚了中国和北美数百位来自知名互联网企业的行业大咖，向寻求就业和技术提升的人群提供直播、录播、面授等多模式教学场景，并通过遍布全国的线下服务中心提供成熟的学习服务，形成完善的“互联网＋教育”解决方案。同时，课工场也为高校、企业、行业提供教育技术赋能，依托 Transform 智能教育生态平台，打造智慧校园、企业大学、行业培训的教育场景，提供一站式教育解决方案。

课工场于 2016 年荣膺新浪网“2016 中国影响力科技创新教育机构”，腾讯网“2016 中国影响力教育品牌”，网易“2016 年度最受信赖教育机构”，小米“2016 教育行业突出贡献奖”。



扫一扫关注课工场公众号

关注微信 立送20币

可购买收费课程



课工场APP客户端下载

产品/设计/开发/运维/运营

随时随地随心学

云计算工程师系列

数据库应用

主　编　肖　睿　于继武

副主编　杨高岭　刘玉强　李　震

中国水利水电出版社

www.waterpub.com.cn

·北京·

内 容 提 要

本书针对具备 Linux 基础的人群，主要介绍了关系型数据库 MySQL、非关系型数据库 MongoDB 的相关知识与应用，以生动详细的理论讲解、深入浅出的实战项目案例，帮助读者掌握现代主流数据库的应用。本书内容包括 MySQL 数据库基础、索引与事务、数据库管理、完全备份与恢复、增量备份与恢复、存储引擎等基础知识，之后的项目案例包括 MySQL 主从复制与读写分离、MongoDB 群集部署。

本书通过通俗易懂的原理讲解及深入浅出的案例分析，并配以完善的学习资源和支持服务，为读者带来全方位的学习体验，包括视频教程、案例素材下载、学习交流社区、讨论组等终身学习内容，更多技术支持请访问课工场 www.kgc.cn。

图书在版编目（CIP）数据

数据库应用 / 肖睿，于继武主编. -- 北京 : 中国水利水电出版社，2017.5
（云计算工程师系列）
ISBN 978-7-5170-5378-1

Ⅰ. ①数… Ⅱ. ①肖… ②于… Ⅲ. ①关系数据库系统 Ⅳ. ①TP311.138

中国版本图书馆CIP数据核字(2017)第099116号

策划编辑：祝智敏　责任编辑：周益丹　加工编辑：夏雪丽　封面设计：梁　燕

书　名	云计算工程师系列 数据库应用　SHUJUKU YINGYONG
作　者	主　编　肖　睿　于继武 副主编　杨高岭　刘玉强　李　震
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路 1 号 D 座　100038） 网　址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电　话：（010）68367658（营销中心）、82562819（万水）
经　售	全国各地新华书店和相关出版物销售网点
排　版	北京万水电子信息有限公司
印　刷	北京泽宇印刷有限公司
规　格	184mm × 260mm　16 开本　12 印张　282 千字
版　次	2017 年 5 月第 1 版　2017 年 5 月第 1 次印刷
印　数	0001—3000 册
定　价	39.00 元

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

版权所有 · 侵权必究

丛书编委会

主　任：肖　睿

副主任：刁景涛

委　员：杨　欢　潘贞玉　张德平　相洪波　谢伟民
庞国广　张惠军　段永华　李　娜　孙　苹
董泰森　曾谆谆　王俊鑫　俞　俊

课工场：李超阳　祁春鹏　祁　龙　滕传雨　尚永祯
张雪妮　吴宇迪　曹紫涵　吉志星　胡杨柳依
李晓川　黄　斌　宗　娜　陈　璇　王博君
刁志星　孙　敏　张　智　董文治　霍荣慧
刘景元　袁娇娇　李　红　孙正哲　史爱鑫
周士昆　傅　峥　于学杰　何娅玲　王宗娟

前　言

“互联网 + 人工智能”时代，新技术的发展可谓一日千里，云计算、大数据、物联网、区块链、虚拟现实、机器学习、深度学习等，已经形成一波新的科技浪潮。以云计算为例，国内云计算市场的蛋糕正变得越来越诱人，以下列举了 2016 年以来该领域发生的部分大事。

（1）中国联通发布云计算策略，并同步发起成立“中国联通沃云 + 云生态联盟”，全面开启云服务新时代。

（2）内蒙古斥资 500 亿元欲打造亚洲最大的云计算数据中心。

（3）腾讯云升级为平台级战略，旨在探索云上生态，实现全面开放，构建可信赖的云生态体系。

（4）百度正式发布“云计算 + 大数据 + 人工智能”三位一体的云战略。

（5）亚马逊 AWS 和北京光环新网科技股份有限公司联合宣布：由光环新网负责运营的 AWS 中国（北京）区域在中国正式商用。

（6）来自 Forrester 的报告认为，AWS 和 OpenStack 是公有云和私有云事实上的标准。

（7）网易正式推出“网易云”。网易将先行投入数十亿人民币，发力云计算领域。

（8）金山云重磅发布“大米”云主机，这是一款专为创业者而生的性能王云主机，采用自建 11 线 BGP 全覆盖以及 VPC 私有网络，全方位保障数据安全。

DT 时代，企业对传统 IT 架构的需求减弱，不少传统 IT 企业的技术人员面临失业风险。全球知名的职业社交平台 LinkedIn 发布报告，最受雇主青睐的十大职业技能中“云计算”名列前茅。2016 年，中国企业云服务整体市场规模超 500 亿元，预计未来几年仍将保持约 30% 的年复合增长率。未来 5 年，整个社会对云计算人才的需求缺口将高达 130 万。从传统的 IT 工程师转型为云计算与大数据专家，已经成为一种趋势。

基于云计算这样的大环境，课工场（kgc.cn）的教研团队几年前就开始策划的“云计算工程师系列”教材应运而生，它旨在帮助读者朋友快速成长为符合企业需求的、优秀的云计算工程师。这套教材是目前业界最全面、专业的云计算课程体系，能够满足企业对高级复合型人才的要求。参与本书编写的院校老师还有于继武、杨高岭、刘玉强、李震等。



课工场是北京大学下属企业北京课工场教育科技有限公司推出的互联网教育平台，专注于互联网企业各岗位人才的培养。平台汇聚了数百位来自知名培训机构、高校的顶级名师和互联网企业的行业专家，面向大学生以及需要“充电”的在职人员，针对与互联网相关的产品设计、开发、运维、推广和运营等岗位，提供在线的直播和录播课程，并通过遍及全国的几十家线下服务中心提供现场面授以及多种形式的教学服务，并同步研发出版最新的课程教材。

除了教材之外，课工场还提供各种学习资源和支持，包括：

- 现场面授课程
- 在线直播课程
- 录播视频课程
- 授课 PPT 课件
- 案例素材下载
- 扩展资料提供
- 学习交流社区
- QQ 讨论组（技术，就业，生活）

以上资源请访问课工场网站 www.kgc.cn。

本套教材特点

（1）科学的训练模式

- 科学的课程体系。
- 创新的教学模式。
- 技能人脉，实现多方位就业。
- 随需而变，支持终身学习。

（2）企业实战项目驱动

- 覆盖企业各项业务所需的 IT 技能。
- 几十个实训项目，快速积累一线实践经验。

（3）便捷的学习体验

- 提供二维码扫描，可以观看相关视频讲解和扩展资料等知识服务。
- 课工场开辟教材配套版块，提供素材下载、学习社区等丰富的在线学习资源。

读者对象

（1）初学者：本套教材将帮助你快速进入云计算及运维开发行业，从零开始逐步成长为专业的云计算及运维开发工程师。

（2）初中级运维及运维开发者：本套教材将带你进行全面、系统的云计算及运维开发学习，逐步成长为高级云计算及运维开发工程师。

课工场出品（kgc.cn）

课程设计说明

课程目标

读者学完本书后，能够掌握数据库相关操作与配置，设计、实施和维护数据库群集。

训练技能

- 掌握 MySQL 数据库的基本配置。
- 理解 MySQL 数据库的备份方式。
- 理解 MySQL 数据库的存储引擎。
- 掌握 MySQL 数据库主从复制、读写分离、多主多从的原理及配置。
- 掌握 MongoDB 数据库的基本配置。
- 掌握 MongoDB 数据库复制集及分片的配置。

设计思路

本书采用了教材 + 扩展知识的设计思路，扩展知识提供二维码扫描，形式为文档、视频等，内容可以随时更新，能够更好地服务读者。课工场网站提供了 Oracle 数据库的视频讲解，感兴趣的读者可登录网站观看。

教材分为 11 个章节、2 个阶段来设计学习，即 MySQL 数据库部分与 MongoDB 数据库部分，具体安排如下：

- 第 1 章～第 8 章介绍 MySQL 数据库基本操作等基础知识，要求掌握 MySQL 数据库备份与恢复、主从复制、读写分离、高可用、多主多从配置。
- 第 9 章～第 11 章介绍 MongoDB 数据库基本操作等基础知识，要求掌握 MongoDB 复制集与分片的相关内容。

章节导读

- 技能目标：学习本章所要达到的技能，可以作为检验学习效果的标准。
- 本章导读：对本章涉及的技能内容进行分析并展开讲解。
- 操作案例：对所学内容的实操训练。
- 本章总结：针对本章内容的概括和总结。
- 本章作业：针对本章内容的补充练习，用于加强对技能的理解和运用。
- 扩展知识：针对本章内容的扩展、补充，对于新知识随时可以更新。

学习资源

- 学习交流社区（课工场）
- 案例素材下载
- 相关视频教程

更多内容详见课工场 www.kgc.cn。



目　录

前言

课程设计说明

第 1 章　MySQL 数据库基础 1

1.1　数据库简介 2

1.1.1　使用数据库的必要性 2

1.1.2　数据库的基本概念 2

1.1.3　数据库发展史 4

1.1.4　关系数据库的基本概念 11

1.2　MySQL 服务基础 14

1.2.1　MySQL 的编译安装 14

1.2.2　访问 MySQL 数据库 17

1.3　使用 MySQL 数据库 19

1.3.1　查看数据库结构 19

1.3.2　创建及删除库和表 21

1.3.3　管理表中的数据记录 22

本章总结 24

本章作业 25

第 2 章　MySQL 索引与事务 27

2.1　数据库索引 28

2.1.1　索引及分类 28

2.1.2　创建及查看索引 30

2.2　MySQL 事务 34

2.2.1　事务的概念及特点 35

2.2.2　事务的操作 35

本章总结 39

本章作业 39

第 3 章　MySQL 数据库管理 41

3.1　用户与授权 42

3.1.1　用户管理 42

3.1.2　授权控制 46

3.2　MySQL 日志管理 50

3.3　数据乱码解决方案 54

本章总结 56

本章作业 56

扩展知识 56

第 4 章　MySQL 完全备份与恢复 57

4.1　MySQL 完全备份 58

4.1.1　数据库备份方式精讲 58

4.1.2　mysqldump 备份演练 60

4.2　MySQL 完全恢复 62

4.2.1　恢复整库操作 62

4.2.2　恢复表操作 65

4.2.3　MySQL 备份思路 67

4.2.4　完全备份恢复案例 67

本章总结 69

本章作业 70

第 5 章　MySQL 增量备份与恢复 71

5.1　MySQL 增量备份 72

5.1.1　增量备份的概念 72

5.1.2　增量备份实战演练 72

5.2　MySQL 增量恢复 75

5.2.1　增量恢复实战 75

5.2.2　企业恢复案例 81

本章总结 82

本章作业 83

第 6 章 MySQL 存储引擎 MyISAM 和 InnoDB..... 85
6.1 MyISAM 存储引擎 86
6.1.1 存储引擎介绍..... 86
6.1.2 MyISAM 的特点..... 87
6.1.3 MyISAM 适用的生产场景举例..... 88
6.2 InnoDB 存储引擎 89
6.2.1 InnoDB 特点介绍 89
6.2.2 InnoDB 适用的生产场景 90
6.2.3 企业选择存储引擎的依据..... 90
6.2.4 配置合适的存储引擎 90
本章总结..... 97
本章作业..... 97

第 7 章 MySQL 主从复制与读写分离 99
7.1 案例分析..... 100
7.1.1 案例概述..... 100
7.1.2 案例前置知识点 100
7.1.3 案例环境..... 102
7.2 案例实施..... 103
7.2.1 搭建 MySQL 主从复制 103
7.2.2 搭建 MySQL 读写分离 106
本章总结..... 111
本章作业..... 111

第 8 章 MySQL 高可用群集 113
8.1 MySQL-MMM 高可用 114
8.1.1 MMM 简介 114
8.1.2 案例环境.....115
8.1.3 案例实施.....115
8.2 MHA 高可用..... 123
本章总结..... 124
本章作业..... 124

第 9 章 MongoDB 基础..... 125
9.1 MongoDB 概述 126
9.1.1 MongoDB 简介 126
9.1.2 MongoDB 的特点..... 126
9.1.3 MongoDB 适用领域 127
9.2 MongoDB 的安装..... 127
9.2.1 安装及运行控制 127
9.2.2 启动和停止 MongoDB..... 128
9.2.3 启动 MongoDB 多实例..... 129
9.3 MongoDB 的存储结构 129
9.3.1 MongoDB 逻辑存储结构..... 130
9.3.2 MongoDB 物理存储结构..... 132
9.3.3 MongoDB 数据类型 133
9.4 MongoDB 基本操作..... 134
9.4.1 在 MongoDB 中创建和删除数据库 .. 135
9.4.2 MongoDB 的集合和文档操作..... 135
9.5 MongoDB 日常维护 136
9.5.1 备份与恢复管理 136
9.5.2 安全管理 139
9.5.3 MongoDB 监控 141
本章总结..... 142
本章作业..... 142

第 10 章 MongoDB 复制集 143
10.1 配置 MongoDB 复制集..... 144
10.1.1 MongoDB 复制集概述..... 144
10.1.2 MongoDB 复制集部署..... 145
10.1.3 MongoDB 复制集切换..... 150
10.2 MongoDB 复制选举原理..... 152
10.2.1 复制集的选举原理..... 153
10.2.2 验证复制集的选举原理 153
10.3 MongoDB 复制集管理 157
10.3.1 配置允许在从节点读取数据 157
10.3.2 查看复制状态信息..... 158
10.3.3 更改 oplog 大小 158
10.3.4 部署认证的复制 159
本章总结..... 161
本章作业..... 161

第 11 章　MongoDB 分片 163
11.1　MongoDB 分片概述 164
11.1.1　什么是分片 164
11.1.2　MongoDB 分片优势 164
11.1.3　MongoDB 分片群集的组成 165
11.2　部署 MongoDB 分片群集 166
11.2.1　分片群集的简单部署 166
11.2.2　实现分片功能 169
11.3　MongoDB 分片管理 172
11.3.1　片键的选择 172
11.3.2　分片的维护 175
本章总结 181
本章作业 181

第1章

MySQL 数据库基础

技能目标

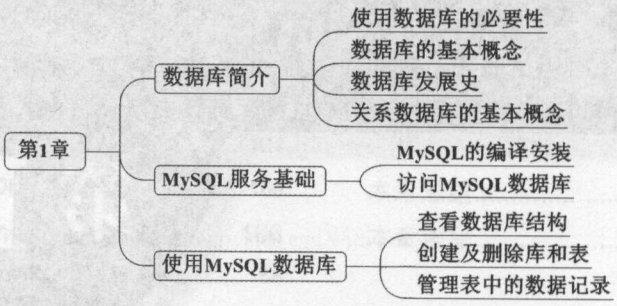
- 理解数据库的基本概念
- 掌握 MySQL 的编译安装
- 会操作 MySQL 数据库

本章导读

21 世纪，人类迈入了“信息爆炸时代”，大量的数据、信息在不断产生，伴随而来的就是如何安全、有效地存储、检索和管理它们。对数据的有效存储、高效访问、方便共享和安全控制已经成为信息时代亟待解决的问题。

知识服务





1.1 数据库简介

1.1.1 使用数据库的必要性

使用数据库可以高效且条理分明地存储数据，使人们能够更加迅速、方便地管理数据。数据库具有以下特点：

- 可以结构化存储大量的数据信息，方便用户进行有效的检索和访问。
- 可以有效地保持数据信息的一致性、完整性，降低数据冗余。
- 可以满足应用的共享和安全方面的要求。

数据库技术是计算机科学的核心技术之一，具有完备的理论基础。对数据库基本概念的掌握，将有助于对数据库的理解。

1.1.2 数据库的基本概念

1. 数据

描述事物的符号记录称为数据（Data）。数据不仅仅包括数字，文字、图形、图像、声音、档案记录等都是数据。

在数据库中，数据是以“记录”的形式按统一的格式进行存储的，而不是杂乱无章的。相同格式和类型的数据统一存放在一起，而不会把“人”和“书”混在一起存储。这样，数据的存储就能够井然有序了。

如图 1.1 中存储的一行数据，在数据库中称为一条“记录”（Record）。每条记录中的每一个输入项称为“列”。图 1.1 中编号、姓名、性别、年龄、民族、专业都是列名。

	编号	姓名	性别	年龄	民族	专业
1	1	王明	男	22	汉	计算机科学与技术
2	2	赵玲	女	21	汉	信息管理
3	3	张雨彤	女	20	汉	音乐
4	4	李新	男	23	回	自动化
5	5	夏雪	女	22	汉	通信

图 1.1 数据记录

2. 数据库和数据库表

不同的记录组织在一起，就形成了数据库（Database，DB）的“表”（Table）。也可以说，表是用来存储具体数据的，如图 1.1 所示。那么数据库和表存在什么关系呢？简单地说，数据库就是表的集合。它是以一定的组织方式存储的相互有关的数据集合。例如，关系数据库的表由记录组成，记录由字段组成，字段由字符或数字组成。它可以供各种用户共享，具有最小冗余度和较高的数据独立性。它是统一管理的相关数据的集合。

通常，数据库并不是简单地存储这些数据的，还要表示它们之间的关系。例如，书和人是存在联系的，书的作者可能就是某个人，因此需要建立书与人的“关系”。这种关系也需要用数据库来表示，因此关系的描述也是数据库的一部分。

3. 数据库管理系统和数据库系统

数据库管理系统（Database Management System，DBMS）是实现对数据库资源有效组织、管理和存取的系统软件。它在操作系统的支持下，支持用户对数据库的各项操作。DBMS 主要包括以下功能：

- 数据库的建立和维护功能：包括建立数据库的结构和数据的录入与转换、数据库的转储与恢复、数据库的重组与性能监视等功能。
- 数据定义功能：包括定义全局数据结构、局部逻辑数据结构、存储结构、保密模式及信息格式等功能。保证存储在数据库中的数据正确、有效和相容，以防止不合语义的错误数据被输入或输出。
- 数据操纵功能：包括数据查询统计和数据更新两个方面。
- 数据库的运行管理功能：这是数据库管理系统的核心部分，包括并发控制、存取控制、数据库内部维护等功能。
- 通信功能：DBMS 其他软件系统之间的通信，如 Access 能与其他 Office 组件进行数据交换。

数据库系统（Database System，DBS）是一个人—机系统，一般由硬件、操作系统、数据库、DBMS、应用软件和数据库用户（包括数据库管理员）组成。用户可以通过 DBMS 操作数据库，也可以通过应用程序操作数据库。

应用程序是利用 DBMS，为解决某个具体的管理或数据处理的任务而编制的一系列命令的有序集合。如果应用程序比较完善，能够提供友好的人机界面，并编译成可执行文件发行，使得普通用户不需要具备计算机的专业知识，在较短时间内就学会使用，那么就称为数据库应用软件。

常用的数据库应用软件有人事管理、财务管理、图书管理等信息管理软件及各类信息咨询系统等。

数据库管理员（Database Administrator，DBA）负责数据库的更新和备份、数据库系统的维护、用户管理等工作，保证数据库系统的正常运行。DBA 一般由业务水平较高、资历较深的人员担任。

注意

数据库、数据库系统、数据库管理系统，甚至数据库表等名词，在日常讨论中通常不严格区别。遇到此情况时，可以根据具体情况，判断出实际所指的是什么。

1.1.3 数据库发展史

1. 数据库系统发展史

数据库技术的发展已经成为先进信息技术的重要组成部分，是现代计算机信息系统和计算机应用系统的基础和核心。数据库技术最初产生于20世纪60年代中期，根据数据模型的发展，可以划分为三个阶段：第一代的网状、层次数据库系统；第二代的关系数据库系统；第三代的以面向对象模型为主要特征的数据库系统。

（1）初级阶段——第一代数据库

自20世纪60年代起，第一代数据库系统问世。它们是层次模型与网状模型的数据库系统，为统一管理和共享数据提供了有力的支撑。在这个阶段中，数据库的代表是1969年IBM公司研制的层次模型的数据库管理系统——IMS（Information Management System，信息管理系统）和20世纪70年代美国数据系统语言协会（CODASYL）下属数据库任务组（DBTG）提议的网状模型。

（2）中级阶段——第二代数据库

20世纪70年代初，第二代数据库——关系数据库开始出现。自1970年IBM研究员德加·考特阐述了关系模型的概念后，IBM大力投入关系数据库的研究。关系数据库的底层实现起来比较容易，所以很快被采用，并进入了众多商业数据库的研发计划。Oracle就是当时顺应关系数据模型的出现而成立的一家专做（关系）数据库的公司。20世纪80年代初，IBM公司的关系数据库系统DB2问世，而Oracle公司也将Oracle移植到桌面计算机上。这时，作为第二代数据库系统的关系数据库，开始逐步取代层次与网状模型的数据库，成为占主导地位的数据库，并成为行业主流。到目前为止，关系数据库系统仍占领数据库应用的主要地位。

关系数据库系统将结构化查询语言（Structured Query Language，SQL）作为数据定义语言（Data Definition Language，DDL）和数据操作语言（Data Manipulation Language，DML），它一诞生就成为关系数据库的标准语言。SQL使得关系数据库中数据库表的查询可以通过简单的、声明性的方式进行，大大简化了程序员的工作。

关系数据库系统构筑在比较高的软件层次上，执行查询的效率普遍偏低。另外，严格的、标准的关系数据库是一个纯理论的模型，如果完全按照关系模型实现，会涉及很多方面的问题，其中一条就是效率不高。在现实环境中，考虑到商业运用的目的，数据库生产厂商各自加入了一些提高效率和提高可用性的功能，舍弃了一些不太现实

的约束。不同的数据库厂商在不同基础上的选择，导致了关系数据库系统向不同方向上的变迁。例如，在这个阶段中，Oracle 加入了“并行”的元素，并开始了向“关系—对象”型数据库的变迁。这样的变迁，也慢慢引出了新一代的数据库系统。

（3）高级阶段——第三代数据库

由于计算机应用的发展，计算机已从传统的科学计算、事务处理等领域，逐步扩展到工程设计、统计、人工智能、多媒体、分布式等领域，这些新的领域需要有新的数据库支撑，而传统关系数据库系统是以商业应用、事务处理为背景而发展起来的，并不完全适用于新领域的应用，因此，需要新的数据库系统，以满足不同领域的要求。

自 20 世纪 80 年代开始，各种适应不同领域的新型数据库系统不断涌现，如工程数据库、多媒体数据库、图形数据库、智能数据库、分布式数据库及面向对象数据库等，特别是面向对象数据库系统，由于其实用性强、适应面广而受到人们的青睐。20 世纪 90 年代后期，形成了多种数据库系统共同支撑应用的局面。当然，在商务应用方面，依然还是关系数据库占主流，不过，已经有一些新的元素被添加进主流商务数据库系统中。例如，Oracle 支持的“关系—对象”数据库模型。

随着科学技术的发展，计算机技术不断应用到各行各业。数据存储需求的不断膨胀，对未来的数据库技术将会有更高的要求。

2. 经典数据模型

数据是现实世界中“量”的抽象，而数据模型（Data Model）是数据特征的抽象。在数据库系统中，数据模型是它的核心与基础。数据模型表现为数据的结构、定义在其上的操作及约束条件。它从概念层次上描述了系统的静态特征、动态特征和约束条件，为数据库系统的信息表示与操作提供了一个抽象框架。

在 DBMS 的发展过程中，出现了网状模型、层次模型和关系模型三种经典的数据模型。由于受限于数学基础、编程技术和硬件条件，最初出现的层次模型和网状模型这两种模型与关系模型相比，与用户接口的上、中层部分更易于实现，所以，这在很长一段时间阻碍了关系模型的发展。

数据模型所描述的内容包括三方面：数据结构、数据操作和数据约束。下面简单介绍三种经典数据模型。

（1）网状模型

1）数据结构。在网状模型中，数据记录组织成图的形式，使用“数据结构图”进行抽象的分析和表示。如图 1.2 所示，该网状模型数据结构图表示的是银行客户、银行账户和银行支行三方面的一种复杂关系。

图 1.2 所示的网状模型抽象于图 1.3 所示的实际情况。它表示了下列几种基本关系：

- 一个银行客户可以拥有多个银行账号（一对多）。
- 一个银行账号也可以被多个客户所有（一对多）。
- 每个银行账户位于特定的银行支行（一对一）。

其中更蕴含着客户和银行之间多对多、多对一的关系。这种复杂的数据关系，在

网状模型上可以得到很好的支持。

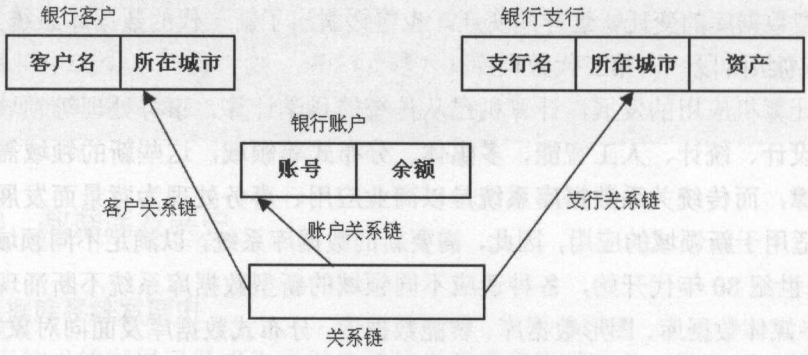


图 1.2　网状模型数据结构

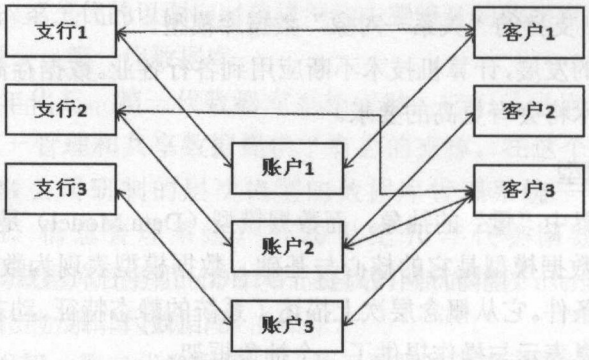


图 1.3　银行、客户、账户关系示例

网状模型适合表达复杂的数据关系的实现，也可以将数据冗余减小到最小。它的数据结构模型能直观反映现实中数据之间的联系。

2）数据操作。从数据结构的定义上不难看出，网状模型的数据操作是建立在关系链基础上的导航式的操作。

针对一个特定的网状模型系统的数据结构，有可能找到最优的查询算法。但是，一旦结构发生变化，就需要新的查询办法。网状模型以图论为基础，还无法得到一个通用的、高效的解决方案。所以，随着关系模型的飞速发展和广泛应用，网状模型已经暂时失去其重要性了。

3）数据约束。网状模型的数据约束是零散孤立的，或者分散在各个节点，或者集中成为一种关系链，这样容易导致不一致性或降低效率。所以，通常网状模型不具体实现数据约束，而由应用程序自身来实现数据约束。这样的情况也使得在网状模型基础上的开发变得困难重重。

（2）层次模型

1）数据结构。层次模型是网状模型的一个特例。在层次模型中，数据记录组织成

树的形式，使用“树结构图”进行抽象的分析和表示，适合一对多的关系模型。图 1.4 所示为企业组织结构的树结构图。

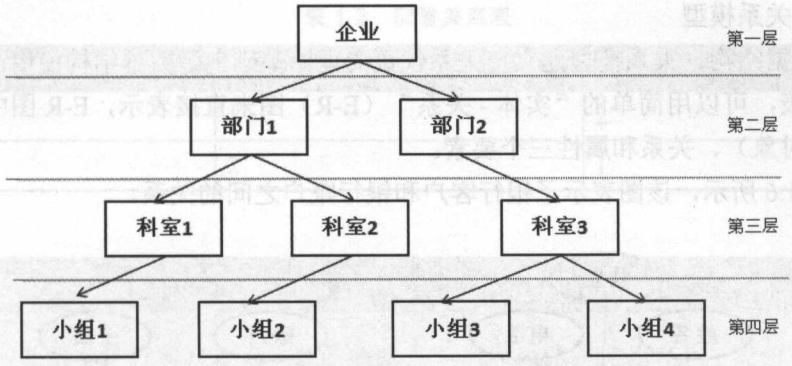


图 1.4　层次模型的树结构图

图 1.4 所示的层次模型的树结构图抽象于图 1.5 所示的实际情况。它包含了下列关系：

- 企业下辖多个部门（一对多）。
- 部门下辖多个科室（一对多）。
- 科室下辖多个小组（一对多）。

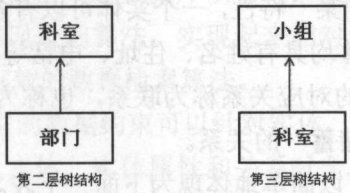


图 1.5　企业组织结构示例

相对于网状模型，层次模型禁止了多对多和多对一的关系，使得它的数据结构相对简单。

2）数据操作。在层次模型上的数据操作不可避免地具有网状模型的特点——导航性。但是，由于禁止了多对一和多对多的关系，因此数据操作相对网状模型而言简单了许多。这样的结构有利于提高数据的查询效率，但数据存取上还存在着必须导航的要求，因此，层次模型的数据库在数据操作上依然比较复杂。

3）数据约束。层次模型的数据约束与网状模型相似，由于结构的简化，去掉了网状模型中多对多和多对一的关系，数据约束处理的复杂性按级数下降，所以，层次模型的数据约束可以做到适当的系统实现，但很多还是依靠应用程序本身实现。

层次模型的实现技术比关系模型优越，比网状模型简单，所以一直独领风骚。它的代表是 IBM 公司的 IMS 系统。该系统曾是使用最早和最广的几个数据库之一，在

历史上曾是最大的数据库之一，由于它的开发者是最早开始处理并发、恢复、完整性和高效查询等问题的人，其中的一些技术和思想自然应用到 DB2 中，这是 DB2 长盛不衰的根源。DB2 有许多性能是非常优秀的，与这段历史不无关系。

（3）关系模型

1）数据结构。关系模型建立在关系代数的理论基础上，数据结构使用简单易懂的二维数据表，可以用简单的“实体 - 关系”（E-R）图来直接表示，E-R 图中包含了实体（数据对象）、关系和属性三个要素。

如图 1.6 所示，该图表示了银行客户和银行账户之间的关系。

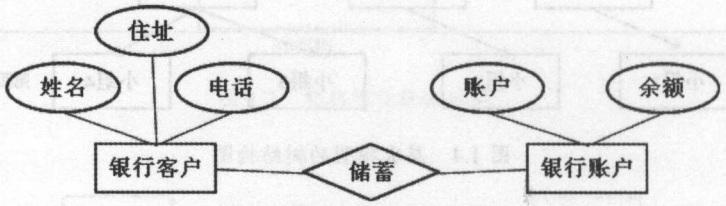


图 1.6　银行客户和银行账户之间的关系

- 实体：也称为实例，对应现实世界中可区别于其他对象的“事件”或“事物”，如银行客户、银行账户等。
- 实体集：具有相同类型及共享相同性质的实体集合。例如，银行所有客户的集合可以定义为“银行客户”实体集。
- 属性：实体所具有的某一特性，一个实体可以有多个属性。例如，“银行客户”实体集中的每个实体均具有姓名、住址、电话等属性。
- 联系：实体集之间的对应关系称为联系，也称为关系。例如，银行客户和银行账户之间存在“储蓄”的关系。

图 1.6 在关系数据库中可以简单地体现为下面三个表。

表 1-1 表示作为银行客户的实体集（客户编码为区分客户实体的唯一标识）。

表 1-1　银行客户表

客户编码	客户姓名	客户住址	客户电话
1	王明	中关村	010-87654321
2	赵玲	回龙观	010-12345678

表 1-2 表示作为银行账户的实体集（账户编码为区分账户实体的唯一标识）。

表 1-2　银行账户表

账户编码	账户	余额
1	23598273959029837	2000.23
2	27298275359025681	5000.55

表 1-3 表示建立银行账户和银行客户之间的储蓄关系（储蓄编码为区分各储蓄关系的唯一标识）。

表 1-3　储蓄关系表

储蓄编码	客户编码	账户编码
1	1	2
2	2	1

注意

虽然在银行客户看来，自己的账户和别人的账户完全不同，是唯一的。但是，银行内部往往还是会使用内部编码来区分管理和服务等不同业务。

2）数据操作。对于数据库的用户而言，关系模型使用从关系代数上抽象出来的数据库操作语言（DML）进行操作。结构化查询语言（SQL）就是其中最重要的一种，已经成为关系数据库的标准操作语言。它的特色是直接面向结果，简化操作步骤，使得数据库应用的设计变得非常简单易懂。

对于数据库的物理结构而言，关系数据库系统的数据结构简单、功能强、数据独立性高、理论基础坚实。严格的关系数据库以二维的数据库表作为基本数据结构，利用简单或复杂的索引技术实现查询算法，实现起来相对比较简单，也方便了预编译技术将 SQL 语言直接转化为有效的数据检索算法。

3）数据约束。关系模型的数据约束可以针对实体，也可以针对实体的属性，还可以针对关系，并可以在定义实体、实体属性和关系时全面实现。关系模型使用的数据定义语言（DDL）和关系模型在理论上对关系数据库核心实现的要求，使得数据约束可以很容易实现，但是它的效率并不高。

综合来说，关系模型相比网状模型和层次模型有更为坚实和完整的理论基础。相比层次模型和网状模型而言，关系模型与用户更靠近些，而网状模型和层次模型与底层实现的结合更紧密。这样的特色也更容易让关系模型成为商业数据库的选择。

3. 当今主流数据库介绍

在数据库技术日益发展的今天，主流数据库代表着成熟的数据库技术。了解常用数据库，就能知道数据库技术发展的程度，以及未来的大体方向。

（1）关系数据库

20 世纪 80 ～ 90 年代是关系数据库产品发展和竞争的时代。在市场逐渐淘汰了第一代数据库管理系统的大局面下，SQL Server、Oracle、IBM DB2、MySQL 等一批很有实力的关系数据库产品走到了主流商用数据库的位置。

1）SQL Server 简介

SQL Server 是 Microsoft 公司的数据库产品，在设计上大量利用了 Microsoft Windows 操作系统的底层结构，直接面向 Microsoft Windows，尤其是 Windows 系列服务器操作系统的用户。

Microsoft Windows 拥有众多的用户群，Microsoft 所有的产品都遵循统一的操作习惯。对数据库基本概念熟悉的 Windows 用户，可以很快地学会使用 SQL Server，上手比较容易。Windows 系统的易用性也让数据库管理员可以更容易、更方便、更轻松地进行管理。

Microsoft 公司针对市场的需求，不断扩展其性能，使得 SQL Server 在网络数据库服务和电子商务方面展示了强大的性能。

2）Oracle 简介

Oracle 公司成立于 1977 年，最初就是专门的数据库公司。

1998 年 9 月，Oracle 公司正式发布 Oracle 8i，“i”代表 Internet，这一版本中添加了大量为支持 Internet 而设计的特性。这一版本为数据库用户提供了全方位的 Java 支持。

在 2001 年 6 月的 Oracle Open World 大会中，Oracle 公司发布了 Oracle 9i，包含应用群集软件 Real Application Clusters（RAC）和商务智能（BI）功能。

2004 年 2 月，Oracle 公司发布了 Oracle 10g 版本，“g”代表 grid（网格）。这一版的最突出特性就是加入了网格计算的功能。

2013 年 6 月，Oracle 12c 正式发布，该版本提供了先进的技术堆栈管理、安全的数据库管理及企业级的服务管理，使企业能够快速实现私有云。

目前，Oracle 数据库成为世界上使用广泛的数据库系统之一。Oracle 公司在数据库领域一直处于领先地位，不仅数据库核心相当优秀，而且其相关的支持产品也相当完善和全面。Oracle 能适应 70 多种操作系统，这也是其他产品难以企及的优势。

3）DB2 简介

1970 年，IBM 公司的一位研究员德加考特发表论文，提出“关系模型”的概念。此后，他被称为“关系数据库之父”。IBM 公司投入巨资，开展包括“SystemR”和“SystemR*”项目在内的关系数据库技术的研究。13 年后，在“SystemR”和“SystemR*”项目的基础上，DB2 以 SystemR 为原型面世。

DB2 支持从 PC 到 UNIX，从中小型机到大型机，从 IBM 到非 IBM（HP 及 Sun UNIX 系统等）的各种操作系统平台。其中，服务器平台可以是 OS/400、AIX、OS/2、HP-UNIX、Sun Solaris 等操作系统，客户机平台可以是 OS/2 或 Windows、DOS、AIX、HP-UX、Sun Solaris 等操作系统。但是，DB2 服务器端的最佳运行环境还是 IBM 自己的操作系统平台 OS/400。

DB2 数据库核心又称为 DB2 通用服务器，可以运行于多种操作系统之上，它根据相应的平台环境做了调整和优化，以便达到较好的性能。由于 IBM 公司在商用服务器领域内的长期优势，在全球 500 强的企业中，超过 80% 的企业曾使用 DB2 作为数据库平台。

4）MySQL 简介

MySQL 也是一个关系型数据库管理系统，现已被 Oracle 公司收购。它与上述大型数据库相比，有不足之处，但是这丝毫没有减少它受欢迎的程度。

MySQL 运行于 Linux 操作系统之上，Apache 和 Nginx 作为 Web 服务器，MySQL 作为后台数据库，PHP/Perl/Python 作为脚本解释器。这四款软件都是免费或开源的，也就是说，企业可以不花一分钱（除人工外）就能建立起一个稳定、高速的网站系统，业内称为“LAMP”组合。因此，其以体积小、速度快、开源等特点，霸占了中小型网站相当大的市场。

（2）非关系数据库

非关系数据库也被称作 NoSQL（Not Only SQL），存储数据不以关系模型为依据，不需要固定的表格式。非关系型数据库作为关系数据库的一个补充，在日益快速发展的网站时代，发挥着高效率与高性能。

非关系型数据库的优点：

- 数据库高并发读写的需求。
- 对海量数据高效率存储与访问。
- 数据库的高扩展性与高可用性的需求。

常用的非关系数据库如 Memcached、Redis、MongoDB、HBase，我们将在后续课程中进行介绍。

1.1.4　关系数据库的基本概念

关系数据库系统是基于关系模型的数据库系统，是关系模型应用到数据库领域的实例化。它的基本概念来自于关系模型。

1. 关系数据库的基本结构

关系数据库使用的存储结构是多个二维表格，即反映事物及其联系的数据描述是以平面表格形式体现的。

在每个二维表中，每一行称为一条记录，用来描述一个对象的信息；每一列称为一个字段，用来描述对象的一个属性。数据表与数据库之间存在相应的关联，这些关联用来查询相关的数据。图 1.7 所示就是一个数据表。

关系数据库是由数据表之间的关联组成的。其中：

- 数据表通常是一个由行和列组成的二维表，每一个数据表分别说明数据库中某一特定的方面或部分的对象及其属性。
- 数据表中的行通常叫做记录或者元组，它代表众多具有相同属性的对象中的一个。
- 数据表中的列通常叫做字段或者属性，它代表相应数据库中存储对象的共有的属性。

2. 主键与外键

（1）主键

数据表中的每行记录都必须是唯一的，而不允许出现完全相同的记录，通过定义主键（主关键字，Primary Key）可以保证记录（实体）的唯一性。

键，即关键字，它是关系模型中一个非常重要的元素。

主键唯一标识表中的行数据，一个主键值对应一行数据。主键由一个或多个字段组成，其值具有唯一性，不允许取空值（NULL）。一个表只能有一个主键。

如果一个属性集能唯一地标识表的一行而又不含有多余的属性，那么这个属性集称为候选键。表中可以有多个候选键，但是只能有一个候选键可以选作表的主键，所有其他候选键称为备用键。例如，在图 1.7 所示的“学生信息统计表”中，“编号”“身份证号”或“姓名”“专业编号”都可以说是候选键，可以将“编号”定义为主键。

DB. stude....学生信息统计表

	编号	姓名	身份证号	性别	年龄	民族	专业编号
	1	王明	110626198801012811	男	22	汉	1
	2	赵玲	146225198906262638	女	21	汉	5
	3	张雨彤	130621199010033522	女	20	汉	3
	4	李新	110265198703051923	男	23	回	2
	5	夏雪	110215198403253562	女	22	汉	2
	6	王林	120026198602092813	男	24	满	4
	7	夏雪	130726199002162828	女	20	汉	5

图 1.7　学生信息统计表

（2）外键

一个关系数据库通常包含多个表，通过外键（Foreign Key）可以使这些表关联起来。

外键是用于建立和加强两个表数据之间的链接的一列或多列。通过将表中主键值的一列或多列添加到另一个表中，可创建两个表之间的链接。这个列就称为第二个表的外键。

例如，在图 1.8 所示的“专业名称表”中，字段“专业编号”是该表的主键，在图 1.7 所示的“学生信息统计表”中也有一个字段“专业编号”，则该字段称为“学生信息统计表”的外键。

DB. stude...bo.专业名称表*

	专业编号	专业
	1	计算机科学与技术
	2	信息管理
	3	自动化
	4	音乐
	5	美术
	6	通信
	7	哲学

图 1.8　专业名称表

在图 1.7 和图 1.8 中，“专业名称表”就称为主表，“学生信息统计表”就称为从表。主表和从表总是成对出现的，相互之间以“外键”形成关联。

3. 数据完整性规则

为了维护数据库中的数据与现实世界的一致性，关系数据库的数据与更新操作必须遵守下列四类完整性规则。

（1）实体完整性规则

实体完整性规则要求关系中的元组在主键的属性上不能有空值。如果出现空值，那么主键值就起不到唯一标识元组的作用。

例如，在图 1.9 所示的“学生信息表”中，每个学生都有一个编号，用来唯一标识每个学生的信息记录，这个编号往往被设为该表的主键，以方便其他数据库表的关联应用。依照实体完整性规则，“编号”字段不允许为空。

DB.stude....学生信息统计表

编号	姓名	身份证号	性别	年龄	民族
1	王明	1106261988010...	男	22	汉
2	赵玲	1462251989062...	女	21	汉
3	张雨彤	1306211990100...	女	20	汉
4	李新	1102651987030...	男	23	回
5	夏雪	1102151984032...	女	22	汉
6	王林	1200261986020...	男	24	满

图 1.9　学生信息表

（2）域完整性规则

域完整性也称列完整性，指定一个数据集对某一个列是否有效或确定是否允许空值。

例如，在图 1.9 所示的“学生信息表”中，定义“性别”字段只能取值为“男”或“女”，这样该列就不会输入其他一些无效的值，如“female”“22”“苗族”等。

（3）引用完整性规则

如果两个表之间相互关联，那么引用完整性规则要求不允许引用不存在的元组。

例如，在图 1.9 所示的表中记录了所有的学生信息，在图 1.10 所示的表中记录了学生的考勤记录。

DB.student - dbo.学生考勤表

序号	姓名	上课时间	下课时间
1	赵玲	2011-05-05 08:00:00.000	2011-05-05 17:00:00.000
2	夏雪	2011-05-09 09:00:00.000	2011-05-09 15:00:00.000
3	王菡	2011-05-09 08:30:00.000	2011-05-09 16:00:00.000

图 1.10　学生考勤表

学生“王菡”在“学生信息表”中不存在，但是在“学生考勤表”中，却有了她的出勤记录，这样的情况是不允许出现的。

（4）用户定义的完整性规则

用户定义的完整性规则是针对某一具体数据的约束条件，由应用环境决定。它反映了某一具体应用所涉及的数据必须满足的语义要求。系统提供定义和检验这类完整性的机制，以便用统一的系统方法进行处理，不再由应用程序承担这项工作。

1.2 MySQL 服务基础

MySQL 是一个真正的多线程、多用户的 SQL 数据库服务，凭借其高性能、高可靠和易于使用的特性，成为服务器领域中最受欢迎的开源数据库系统。在 2008 年以前，MySQL 项目由 MySQL AB 公司进行开发、发布和支持，之后历经 Sun 公司收购 MySQL AB 公司，Oracle 公司收购 Sun 公司的过程，目前 MySQL 项目由 Oracle 公司负责运营和维护。

本节将介绍 MySQL 的编译安装过程、服务控制方法，以及如何使用客户端工具访问 MySQL 数据库。

1.2.1 MySQL 的编译安装

为了确保 MySQL 数据库功能的完整性、可定制性，本小节将采用源代码编译的方式安装 MySQL 数据库系统。之前在学习 LAMP 时介绍过 MySQL 5.5 的编译安装，本节将选用 mysql-5.7.17.tar.gz，其官方站点为 https://dev.mysql.com/。

1. 准备工作

（1）如果 CentOS 7.3 已安装 mariadb 数据库，必须先将其卸载，然后再开始进行新下载的源码安装。

```
[root@www ~]# rpm -q mariadb
 mariadb-5.5.52-1.el7.x86_64

[root@www ~]# yum remove mariadb
[root@www ~]# yum install gcc gcc-c++ ncurses-devel
```

（2）MySQL 5.7 源码安装需要 cmake 编译安装，所以先下载最新稳定版本的 cmake 包并安装，官方下载网站：https://cmake.org/download/。

```
[root@www ~]# tar zxf  cmake-3.7.2.tar.gz
[root@www ~]# cd cmake-3.7.2
[root@www cmake-3.7.2]# ./configure
[root@www cmake-3.7.2]# gmake && gmake install
```

（3）MySQL5.7 需要 Boost 这个库，所以也需要下载安装，这里下载 1_59_0 版本，注意这个版本和 MySQL 的版本是相对应的。注意在 cmake 的时候不添加 -DWITH_BOOST 这个参数时它会报错告诉你需要 Boost 的哪个版本。

官方下载网站：http://sourceforge.net/projects/boost/files/boost/

```
[root@www ~]# tar zxf boost_1_59_0.tar.gz
[root@www ~]# mv boost_1_59_0/ /usr/local/boost
```

2. 源码编译及安装

（1）创建运行用户

为了加强数据库服务的权限控制，建议使用专门的运行用户，如 mysql。此用户不需要直接登录到系统，可以不创建宿主文件夹。

```
[root@www ~]# groupadd  mysql
[root@www ~]# useradd -M -s /sbin/nologin mysql -g mysql
```

（2）解包

将下载的 mysql 源码包解压，释放到 /usr/src 目录下，并切换到展开后的源码目录。

```
[root@www ~]# tar zxf mysql-5.7.17.tar.gz -C /usr/src
[root@www ~]# cd /usr/src/mysql-5.7.17
```

（3）配置

在内容丰富、结构庞大的企业网站平台中，可能会用到多种字符集的网页，相应地数据库系统也应该支持不同的字符集编码。在配置过程中，可以将默认使用的字符集设置为 utf8，并添加其他字符集的支持。

```
[root@www mysql-5.7.17]# cmake  -DCMAKE_INSTALL_PREFIX=/usr/local/mysql
-DSYSCONFDIR=/etc  -DDEFAULT_CHARSET=utf8 -DDEFAULT_COLLATION=utf8_general_ci
-DWITH_EXTRA_CHARSETS=all -DWITH_BOOST=/usr/local/boost
```

上述配置命令中，各选项的含义如下：

- DCMAKE_INSTALL_PREFIX：指定将 mysql 数据库程序安装到某目录下，如目录 /usr/local/ mysql。
- DSYSCONFDIR：指定初始化参数文件目录。
- DDEFAULT_CHARSET：指定默认使用的字符集编码，如 utf8。
- DDEFAULT_COLLATION：指定默认使用的字符集校对规则，utf8_general_ci 是适用于 UTF-8 字符集的通用规则。
- DWITH_EXTRA_CHARSETS：指定额外支持的其他字符集编码。
- DWITH_BOOST：指定 Boost 库的位置，5.7 版本必须添加这个参数。

（4）编译并安装

```
[root@www mysql-5.7.17]# make
[root@www mysql-5.7.17]# make install
```

3. 安装后的其他调整

（1）对数据库目录进行权限设置

```
[root@www ~]# chown -R mysql:mysql /usr/local/mysql
```

（2）建立配置文件

在 MySQL 源码目录中的 support-files 文件夹下，根据以下参考内容建立 MySQL 系统的 /etc/my.cnf 配置文件。

```
[root@www ~]# rm -rf /etc/my.cnf   // 如果原来 etc 文件夹下有 my.cnf 文件可以删除
[root@www ~]#
cp /usr/local/mysql/support-files/my-default.cnf /etc/my.cnf
```

（3）初始化数据库

为了能够正常使用 MySQL 数据库系统，应以运行用户 mysql 的身份执行初始化，指定数据存放目录等。注意最后一行会生成一个随机的临时密码，请牢记！

```
[root@www mysql-5.7.17]#
/usr/local/mysql/bin/mysqld --initialize --user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/
        mysql/data
2017-02-16T08:19:09.867068Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is
        deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation
        for more details).
2017-02-16T08:19:15.264290Z 0 [Warning] InnoDB: New log files created, LSN=45790
2017-02-16T08:19:15.887558Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
2017-02-16T08:19:16.032143Z 0 [Warning] No existing UUID has been found, so we assume that
        this is the first time that this server has been started. Generating a new UUID: 9b23c0fc-f420-
        11e6-8562-44a8424bf8c2.
2017-02-16T08:19:16.050948Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_
        executed' cannot be opened.
2017-02-16T08:19:16.052183Z 1 [Note] A temporary password is generated for root@localhost: ;kL;
        lnmbu9oA   // 初始 root 用户密码
```

（4）设置环境变量

为了方便在任何目录下使用 mysql 命令，需要在 /etc/profile 设置环境变量。

```
[root@www mysql-5.7.17]# echo "export PATH=$PATH:/usr/local/mysql/bin" >> /etc/ profile
[root@www mysql-5.7.17]# . /etc/profile        // 立即生效
```

4. 添加系统服务

若希望添加 mysqld 系统服务，以便通过 systemctl 进行管理，可以直接使用源码包中提供的服务脚本。找到 support-files 文件夹下的 mysql.server 脚本文件，将其复制到 /etc/rc.d/init.d 目录下，并改名为 mysqld，然后再设置执行权限。

```
[root@www mysql-5.7.17]# cp support-files/mysql.server /etc/rc.d/init.d/ mysqld
[root@www mysql-5.7.17]# chmod +x /etc/rc.d/init.d/mysqld
```

将 MySQL 添加为 systemd 标准服务，方便以后使用“systemctl”命令进行管理。

```
[root@www ~]# vi /lib/systemd/system/mysqld.service
[Unit]
Description=mysqld
After=network.target

[Service]
Type=forking
ExecStart=/etc/rc.d/init.d/mysqld start
```

```
ExecReload=/etc/rc.d/init.d/mysqld restart
ExecStop=/etc/rc.d/init.d/mysqld stop

PrivateTmp=true

[Install]
WantedBy=multi-user.target
[root@www ~]# systemctl daemon-reload
[root@www ~]# systemctl enable mysqld
[root@www ~]# systemctl start mysqld
```

这样，以后就可以使用 systemctl 工具或直接调用 /etc/init.d/mysqld 脚本来控制 MySQL 数据库服务了。例如，若要启动 mysqld 服务，并查看其运行状态，可以执行以下操作。

```
[root@www ~]# systemctl start mysqld.service
[root@www ~]# systemctl status mysqld.service
mysqld.service - mysqld
  Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; vendor preset: disabled)
  Active: active (running) since Sun 2017-02-19 19:43:37 CST; 2min 58s ago
 Process: 36550 ExecStart=/etc/rc.d/init.d/mysqld start (code=exited, status=0/SUCCESS)
Main PID: 36555 (mysqld_safe)
  CGroup: /system.slice/mysqld.service
          ├─ 36555 /bin/sh /usr/local/mysql/bin/mysqld_safe --datad...
          └─ 36659 /usr/local/mysql/bin/mysqld --basedir=/usr/local...

Feb 19 19:43:34 www systemd[1]: Starting mysqld...
Feb 19 19:43:35 www mysqld[36550]: Starting MySQL.Logging to '/usr....
Feb 19 19:43:37 www mysqld[36550]: . SUCCESS!
Feb 19 19:43:37 www systemd[1]: Started mysqld.
Hint: Some lines were ellipsized, use -l to show in full.
[root@www ~]# netstat -anpt | grep mysqld

tcp6       0      0 :::3306                 :::*                    LISTEN      133192/mysqld
```

MySQL 服务器默认通过 TCP 3306 端口提供服务。通过编辑 /etc/my.cnf 配置文件中 [mysqld] 配置段的“port = 3306”行，可以更改监听端口。

1.2.2　访问 MySQL 数据库

MySQL 数据库系统也是一个典型的 C/S（客户端 / 服务器）架构的应用，要访问 MySQL 数据库需要使用专门的客户端软件。在 Linux 系统中，最简单、易用的 MySQL 客户端软件是其自带的 mysql 命令工具。

1. 登录到 MySQL 服务器

经过安装后的初始化过程，MySQL 数据库的默认管理员用户名为“root”，密码为给定的随机密码。以 root 用户登录本机的 MySQL 数据库，可以执行以下操作。

```
[root@www ~]# mysql  -u root -p
Enter password:          // 根据提示输入上述随机的密码，然后修改密码。
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 5.7.17

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set password =password('h@O123.com');
Query OK, 0 rows affected (0.00 sec)
```

MySQL 5.7 版本默认启用了密码增强插件 validate_password，新密码必须符合密码复杂性要求，如果在测试环境中，可以禁用此插件。

2. 执行 MySQL 操作语句

验证成功以后将会进入提示符为“mysql> ”的数据库操作环境，用户可以输入各种操作语句对数据库进行管理。每一条 MySQL 操作语句以分号“;”表示结束，输入时可以不区分大小写，但习惯上将 MySQL 语句中的关键字部分大写。

例如，以用户名 root 登录到“mysql>”环境后，执行“SHOW DATABASES;”语句可以查看当前数据库中有哪些库。

```
[root@www ~]# mysql -u root –p123456

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

3. 退出“mysql>”操作环境

在“mysql> ”操作环境中，执行“EXIT”或“QUIT”命令可以退出 mysql 命令工具，返回原来的 Shell 环境。

```
mysql> EXIT
Bye
[root@www ~]#
```

1.3　使用 MySQL 数据库

熟悉安装及访问 MySQL 数据库以后，接下来将学习使用 MySQL 数据库的基本操作，这也是在服务器运维工作中不可或缺的知识。本节中的所有数据库语句均在“mysql> ”操作环境中执行。

1.3.1　查看数据库结构

下面分别介绍查看数据库、表结构的相关操作语句。

1. 查看当前服务器中包含的库

SHOW DATABASES 语句：用于查看当前 MySQL 服务器中包含的库。经初始化后的 MySQL 服务器，默认建立了四个库：sys、mysql、information_schema 和 performance_schema（其中 mysql 库中包含了用户认证相关的表），执行以下操作可以进行查看。

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

2. 查看当前使用的库中包含的表

SHOW TABLES 语句：用于查看当前所在的库中包含的表。在操作之前，需要先使用 USE 语句切换到所使用的库。例如，执行以下操作可以显示 mysql 库中包含的所有表。

```
mysql> USE mysql;
Database changed
```

```
mysql> SHOW TABLES;
+--------------------------+
| Tables_in_mysql          |
+--------------------------+
| columns_priv             |
| db
| engine_cost              |
| event                    |
……        // 省略部分内容
| user                     |
+--------------------------+
31 rows in set (0.01 sec)
```

MySQL 数据库的数据文件存放在 /usr/local/mysql/data 目录下，每个数据库对应一个子目录，用于存储数据表文件。每一个数据表对应为三个文件，后缀名分别为“.frm”“.myd”和“.myi”。当然也有少数以 opt、csm、csv、ibd 结尾的。

3. 查看表的结构

DESCRIBE 语句：用于显示表的结构，即组成表的各字段（列）的信息。需要指定“库名 . 表名”作为参数；若只指定表名参数，则需先通过 USE 语句切换到目标库。例如，执行以下操作可以查看 mysql 库中的 user 表的结构，与直接执行“DESCRIBE mysql.user;”语句的效果相同。

```
mysql> USE mysql;
Database changed
mysql> DESCRIBE user;
+-----------------+-----------------+------+-----+---------+-------+
| Field           | Type            | Null | Key | Default | Extra |
+-----------------+-----------------+------+-----+---------+-------+
| Host            | char(60)        | NO   | PRI |         |       |
| User            | char(16)        | NO   | PRI |         |       |
| Select_priv     | enum('N','Y')   | NO   |     | N       |       |
……      // 省略部分内容
+-----------------+-----------------+------+-----+---------+-------+
45 rows in set (0.00 sec)
```

数据库目前标准的指令集是 SQL。SQL 是 Structured Query Language 的缩写，即结构化查询语言。它是 1974 年由 Boyce 和 Chamberlin 提出来的，1975—1979 年，IBM 公司研制的关系数据库管理系统原型 System R 实现了这种语言，经过多年的发展，SQL 语言得到了广泛的应用。

SQL 语言主要由以下几部分组成：

- DDL（Data Definition Language，数据定义语言）：用来建立数据库、数据库对象和定义其列，如 CREATE、ALTER、DROP。
- DML（Data Manipulation Language，数据操纵语言）：用来查询、插入、删

除和修改数据库中的数据，如 SELECT、INSERT、UPDATE、DELETE。

- DCL（Data Control Language，数据控制语言）：用来控制数据库组件的存取许可、存取权限等，如 COMMIT、ROLLBACK、GRANT、REVOKE。

1.3.2　创建及删除库和表

当需要为网站平台提供数据库服务时，如何操作来创建新的库呢？根据一份二维数据表格，如何在现有的库中创建表？对于废弃的数据表和库，又该如何进行删除？本小节主要解决这三个问题。

1. 创建新的库

CREATE DATABASE 语句：用于创建一个新的库，需指定数据库名称作为参数。例如，执行以下操作可以创建一个名为 auth 的库。

```
mysql> CREATE DATABASE auth;
Query OK, 1 row affected (0.01 sec)
```

刚创建的数据库是空的，其中不包含任何表，在 /usr/local/mysql/data 目录下会自动生成一个与新建的库名相同的空文件夹。

2. 创建新的表

CREATE TABLE 语句：用于在当前库中创建新的表，需指定数据表名称作为参数，并定义该表格所使用的各字段。基本格式如下所示：

```
CREATE TABLE 表名 ( 字段 1 名称 类型 , 字段 2 名称 类型 , …, PRIMARY KEY ( 主键名 ))
```

创建表之前，应先明确数据表格的结构、各字段的名称和类型等信息。例如，若要创建一个包含用户名、密码字串的用户验证表（表 1-4），应先分析表格结构。

表 1-4　用户验证表的内容

用户名	密码字串
zhangsan	*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9
lisi	*2A032F7C5BA932872F0F045E0CF6B53CF702F2C5
wangwu	*FF8DAB1D945848EB3339D5E583B3FFF7F28AA631

上述表格中，“用户名”为不超过 16 个字符的字符串，且不能为空；“密码字串”为不超过 48 个字符的字符串（插入记录时使用 MySQL 的函数加密），默认值为空字串。考虑到字符集的兼容性，最好不要使用中文字段名，改用 user_name、user_passwd 分别表示用户名、密码字串。表格中不能拥有同名的用户，因此可将 user_name 作为主键。

针对上述分析结果，可以在 auth 库中按如下操作创建 users 表。其中，字段定义部分的 DEFAULT 语句用于设置默认的密码字串，PRIMARY 语句用于设置主键字段名。

```
mysql> USE auth;
mysql> CREATE TABLE users (user_name CHAR(16) NOT NULL, user_passwd CHAR(48)
    DEFAULT '', PRIMARY KEY (user_name));
Query OK, 0 rows affected (0.00 sec)
```

3. 删除一个数据表

DROP TABLE 语句：用于删除库中的表，需要指定“库名 . 表名”作为参数；若只指定表名参数，则需先通过执行“USE”语句切换到目标库。例如，执行以下操作可以删除 auth 库中的 users 表。

```
mysql> DROP TABLE auth.users;
Query OK, 0 rows affected (0.00 sec)
```


4. 删除一个数据库

DROP DATABASE 语句：用于删除指定的库，需要指定库名作为参数。例如，执行以下操作可以删除名为 auth 的库。

```
mysql> DROP DATABASE auth;
Query OK, 0 rows affected (0.01 sec)
```

1.3.3 管理表中的数据记录

参考上节中的步骤重新创建 auth 库和 users 表，下面将以 users 表为基础，学习向表中插入、查询、修改及删除数据记录的操作。

1. 插入数据记录

INSERT INTO 语句：用于向表中插入新的数据记录。语句格式如下所示：

```
INSERT INTO 表名 ( 字段 1, 字段 2,…) VALUES( 字段 1 的值 , 字段 2 的值 ,…)
```

执行以下操作将会向 auth 库中的 user 表插入一条记录：用户名为“zhangsan”，对应的密码为“123456”。注意：VALUES 部分的值应与前面指定的各字段逐一对应。

```
mysql> use auth;
Database changed
mysql> INSERT INTO users(user_name,user_passwd) VALUES('zhangsan', PASSWORD (
        '123456'));

Query OK, 1 row affected, 1 warning (0.19 sec)
```

在插入新的数据记录时，如果这条记录完整包括表中所有字段的值，则插入语句中指定字段的部分可以省略。例如，执行以下操作也可以向 auth 库中的 user 表插入一条新的记录：用户名为“lisi”，对应的密码为“654321”。

```
mysql> INSERT INTO users VALUES('lisi', PASSWORD('654321'));
```

```
Query OK, 1 row affected, 1 warning (0.19 sec)
```

2. 查询数据记录

SELECT 语句：用于从指定的表中查找符合条件的数据记录。MySQL 数据库支持标准的 SQL 查询语句，语句格式如下所示：

```
SELECT 字段名 1, 字段名 2,… FROM 表名 WHERE 条件表达式
```

表示所有字段时，可以使用通配符“*”，若要显示所有的数据记录则可以省略 WHERE 条件子句。例如，执行以下操作可以查看 auth 库中 users 表内的所有数据记录，其中密码字串因已加密，因此不会直接显示出实际的密码内容。

```
mysql> select * from auth.users;
+-----------+-------------------------------------------+
| user_name | user_passwd                               |
+-----------+-------------------------------------------+
| lisi      | *2A032F7C5BA932872F0F045E0CF6B53CF702F2C5 |
| zhangsan  | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------+-------------------------------------------+
2 rows in set (0.00 sec)
```

当需要根据特定的条件查找记录时，WHERE 条件子句则是必不可少的。例如，若要查找 users 表中用户名为“zhangsan”的记录，显示其中用户名、密码字段的信息，可以执行以下操作。

```
mysql> SELECT user_name,user_passwd FROM auth.users where user_name= 'zhangsan';
+-----------+-------------------------------------------+
| user_name | user_passwd                               |
+-----------+-------------------------------------------+
| zhangsan  | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------+-------------------------------------------+
1 row in set (0.00 sec)
```

3. 修改数据记录

UPDATE 语句：用于修改、更新表中的数据记录。语句格式如下所示：

```
UPDATE 表名 SET 字段名 1= 字段值 1[, 字段名 2= 字段值 2]  WHERE 条件表达式
```

执行以下操作可以修改 users 表中用户名为“lisi”的记录，将密码字串设为空值。验证记录内容可以发现 lisi 用户的密码串值已变为空白。

```
mysql> UPDATE auth.users SET user_passwd=PASSWORD('') WHERE user_name= 'lisi';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> select * from auth.users;
+-----------+-------------------------------------------+
| user_name | user_passwd                               |
```

```
+-----------+-------------------------------------------+
| lisi      |                                           |
| zhangsan  | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------+-------------------------------------------+
2 rows in set (0.00 sec)
```

若是在 Linux 命令行环境中执行，还可以使用 mysqladmin 工具来设置密码，但是会报一些警告。例如，执行以下操作并验证原密码后，可将数据库用户 root 的密码设置为“123457”。

```
[root@www ~]# mysqladmin -u root -p password '123457'
Enter password:
mysqladmin: [Warning] Using a password on the command line interface can be insecure.
Warning: Since password will be sent to server in plain text, use ssl connection to ensure password safety.
```


4. 删除数据记录

DELETE 语句：用于删除表中指定的数据记录。语句格式如下所示：

```
DELETE FROM 表名 WHERE 条件表达式
```

执行以下操作可以删除 users 表中用户名为“lisi”的数据记录，验证记录内容可以发现 lisi 用户的数据记录已经消失。

```
mysql> DELETE FROM auth.users WHERE user_name='lisi';
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM auth.users;
+-----------+-------------------------------------------+
| user_name | user_passwd                               |
+-----------+-------------------------------------------+
| zhangsan  | *6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
+-----------+-------------------------------------------+
1 row in set (0.00 sec)
```

在 MySQL 5.7 版本数据库服务器中，将 MySQL 库里面 user 表中的 password 字段修改为 authentication_string 字段，已经删除了一些空用户，提高了安全性。

```
mysql> SELECT user,host,authentication_string FROM mysql.user;
+-----------+-----------+-------------------------------------------+
| user      | host      | authentication_string                     |
+-----------+-----------+-------------------------------------------+
| root      | localhost | *F13F0EEE74714A1A9922D61FC15789AD75FE4958 |
| mysql.sys | localhost | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
+-----------+-----------+-------------------------------------------+
2 rows in set (0.00 sec)
```

本章总结

- 数据库由数据库表和其他数据对象组成。

- 经典数据模型有网状模型、层次模型和关系模型。
- 主键由一个或多个字段组成，其值具有唯一性，而且不允许取空值（NULL）。一个表只能有一个主键。
- 一个关系数据库通常包含多个表，可以通过外键将这些表关联起来。
- MySQL 是一个开源的 SQL 数据库软件，默认使用 TCP 3306 端口提供服务，配置文件是 /etc/my.cnf。
- MySQL 的基本管理操作包括查看数据库结构、创建及删除库和表、管理表中的数据记录。

本章作业

1．什么是主键、外键？

2．在编译安装 MySQL 数据库系统时，如何添加对多种字符集的支持？

3．已知 MySQL 服务器的 IP 地址为 192.168.1.241，监听的端口为 3456，授权访问的用户为 jerry，则使用 MySQL 客户端程序应如何进行连接？（提示：-P 选项指定端口；-h 选项指定目标主机地址）

4．新建 kgc 库，并在 kgc 库中创建一份 product 表（见表 1-5），录入数据并查询产品代号中包含“dev”的记录内容。

表 1-5　product 表的内容

产品代号	技术方向	培养目标
kgcdevj	Java 开发	精通 JavaEE 平台开发的软件工程师
kgcdevbd	大数据开发	精通大数据平台开发的软件工程师
kgctest	软件测试	精通系统和软件测试的测试工程师
kgcyun	云计算	精通自动化运维、云计算运维的网络工程师

随手笔记



第2章

MySQL 索引与事务

技能目标

- 理解索引的概念并掌握索引的创建
- 理解事务的概念并掌握事务的操作

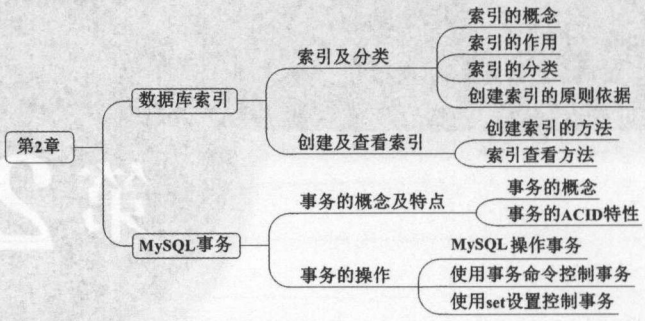
本章导读

在企业信息化的过程中，数据库中表的数据量越来越大，性能会急剧下降，创建索引对于保持良好的性能非常关键。索引是对查询性能优化最有效的手段，能够轻易将查询性能提高几个数量级。

执行数据库操作时，有时需要多个操作表示一个操作流程，要么都执行，要么都不执行，这就是数据库的事务。本章将介绍如何创建 MySQL 的索引和事务。

知识服务





2.1 数据库索引

数据库索引是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中的数据。

2.1.1 索引及分类

1. 索引的概念

索引是一种特殊的文件，包含着对数据表中所有记录的引用指针。更通俗地说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度。例如需要遍历 200 条数据，在没有索引的情况下，数据库会遍历全部 200 条数据后选择符合条件的；而有了相应的索引之后，数据库会直接在索引中查找符合条件的选项。数据库索引就是为了提高表的搜索效率而对某些字段中的值建立的目录。

2. 索引的作用

建立索引的目的是加快对表中记录的查找或排序。为表设置索引要付出代价：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间（因为索引也要随之变动）。但是相比付出的代价，索引的作用显得更为重要：

（1）设置了合适的索引之后，数据库利用各种快速的定位技术，可以大大加快数据的查询速度，这也是创建索引最主要的原因。

（2）当表很大时，或者查询涉及多个表时，使用索引可使查询速度加快成千倍。

（3）可以降低数据库的 IO 成本，并且索引还可以降低数据库的排序成本。

（4）通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

（5）可以加快表和表之间的连接。

（6）在使用分组和排序子句进行数据查询时，可以显著减少查询中分组和排序的时间。

3. 索引的分类

MySQL 的索引分为以下几类：

（1）普通索引，这是最基本的索引类型，而且它没有唯一性之类的限制。

（2）唯一性索引，这种索引和前面的“普通索引”基本相同，但有一个区别：索引列的所有值只能出现一次，即必须唯一。当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在 employee 表中职员的姓（name）上创建了唯一索引，则任何两个员工都不能同姓。

（3）主键索引，主键是一种唯一性索引，但它必须指定为“PRIMARY KEY”。在数据库中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。

（4）全文索引，索引类型为 FULLTEXT，全文索引可以在 CHAR、VARCHAR 或者 TEXT 类型的列上创建。

（5）单列索引与多列索引，索引可以是单列上创建的索引，也可以是在多列上创建的索引。多列索引可以区分其中一列可能有相同值的行。如果经常同时搜索两列或多列或按两列或多列排序时，索引也很有帮助。例如，如果经常在同一查询中为姓和名两列设置查询条件，那么在这两列上创建多列索引将很有意义。

4. 创建索引的原则依据

索引可以提升数据库查询的速度，但并不是任何情况都需要创建索引。因为索引本身会消耗系统资源，更重要的是在有索引的情况下，数据库查询会先进行索引查询，然后定位到具体的数据行，如果索引使用不当，反而会增加数据库的负担，下面列出了创建索引的原则依据。

（1）表的主键、外键必须有索引。主键具有唯一性，索引值也是唯一的，查询时可以快速定位到数据行。外键一般关联的是另一个表的主键，所以在多表查询时也可以快速定位。

（2）数据量超过 300 行的表应该有索引。数据量较大时，如果没有索引，需要把表遍历一遍，严重影响数据库的性能。

（3）经常与其他表进行表连接的表，在连接字段上应该建立索引。

（4）唯一性太差的字段不适合建立索引。如果索引字段的数据唯一性太差，是不适合创建的。比如某表有 10000 行某个字段是“课工场”，那么它们的索引值是相同的，如果创建索引查询，数据库需要匹配这 10000 行的索引值，然后再定位数据表中的数据，并不能提升查询速度，反而会变慢。

（5）更新太频繁的字段不适合创建索引。在表中进行增加、删除、修改操作时，索引也有相应操作产生。字段更新得过于频繁，对于系统资源占用也会更多。

（6）经常出现在 Where 子句中的字段，特别是大表的字段，应该建立索引。

（7）索引应该建在选择性高的字段上。如果很少的字段拥有相同值，即有很多独特值，则选择性很高。

（8）索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引。

2.1.2 创建及查看索引

在 MySQL 中，可以使用命令创建多种类型的索引，也可以查看索引。

1. 创建索引的方法

根据企业需求选择了合适的索引之后，可以使用 CREATE INDEX 创建索引，加上各个关键字，便可创建各个类型的索引。

（1）创建普通索引，命令格式是：

```
CREATE INDEX < 索引的名字 > ON tablename（列的列表）
```

首先，确保数据库 imployee_salary 及数据表 IT_salary 已经创建。

```
mysql> use imployee_salary;
Database changed
mysql> show tables;
+--------------------------+
| Tables_in_imployee_salary |
+--------------------------+
| IT_salary                |
+--------------------------++--------------------+
```

然后，确保 IT_salary 中的数据已经添加。

```
mysql> select * from IT_salary;
+----------------------+----------+--------+----------+--------+--------+--------+
| 岗位类别   | 姓名    | 年龄    | 员工 ID | 学历    | 年限    | 薪资    |
+----------------------+----------+--------+----------+--------+--------+--------+
| 网络工程师          | 孙空武 | 27      |          11 | 本科 |    3 |  4800 |
| Windows 工程师     | 蓝凌    | 19      |          12 | 中专 |    2 |  3500 |
| Linux 工程师       | 姜纹    | 32      |          13 | 本科 |    8 | 15000 |
| Java 工程师        | 关园    | 38      |          14 | 大专 |   10 | 16000 |
| 硬件驱动工程师      | 罗中昆 | 29      |          15 | 大专 |    9 | 16500 |
+----------------------+----------+--------+----------+--------+--------+--------+
5 rows in set (0.01 sec)            |
```

使用索引命令创建“薪资”字段的普通索引，索引的名字是 salary_index，为了使索引的名字容易识别，建议名字以 _index 结尾。

```
mysql> create index salary_index on IT_salary( 薪资 );
```

使用 show index 命令可以查看创建的索引，后面章节会讲解此命令，这里只演示使用。

```
mysql> show index from IT_salary;
| Table     | Non_unique | Key_name     | Seq_in_index | Column_name |
| IT_salary |          1 | salary_index |            1 | 薪资        |
```

Key_name 对应的是索引名字，显示名字是 salary_index 的索引已经存在。Non_

unique 对应值是 1，表示不是唯一性索引。

（2）创建唯一性索引，命令格式是：

```
CREATE UNIQUE INDEX < 索引的名字 > ON tablename（列的列表）
```

该索引比普通索引多了一个 UNIQUE 关键字，下面对字段“姓名”创建唯一性索引。

```
mysql> create unique index salary_unique_index on IT_salary( 姓名 );
```

使用 show index 命令查看创建的索引。

```
mysql> show index from IT_salary;
| Table     | Non_unique | Key_name     | Seq_in_index | Column_name |
| IT_salary |          0 | salary_unique_index |            1 | 姓名        |
```

名字为 salary_unique_index 的索引已经创建，它的 Non_unique 值是 0，表示是唯一性索引。

（3）创建主键索引，有两种方式，一种是在创建表的同时创建主键，主键索引自动创建，命令格式如下：

```
CREATE TABLE tablename ([...],PRIMARY KEY( 列的列表 ));
```

另一种是已经创建了表，没有指定主键，然后修改表加入主键，主键索引会自动创建，命令格式如下：

```
ALTER TABLE tablename ADD PRIMARY KEY( 列的列表 );
```

下面使用 show index 命令查看主键索引：

```
mysql> show index from IT_salary;
| Table     | Non_unique | Key_name     | Seq_in_index | Column_name |
| IT_salary |          0 | PRIMARY      |            1 | 员工 ID     |
```

这里需要注意，主键索引是设置主键后自动创建，并没有指定名字，系统自动生成名字 PRIMARY。

（4）在 MySQL 中使用全文索引（FULLTEXT index），目前只有使用 MyISAM 类型表的时候有效（MyISAM 是默认的表类型）。全文索引可以建立在 TEXT、CHAR 或者 VARCHAR 类型的字段或者字段组合上，创建表时指定或修改表时指定全文索引，命令格式如下：

```
CREATE TABLE  表名 ( 列名 TEXT,FULLTEXT( 列名 )) engine=MyISAM;
ALTER TABLE 表名 ADD FULLTEXT ( 列名 );
mysql> CREATE TABLE IT_desc(descript TEXT,FULLTEXT(descript)) engine=MyISAM;
```

使用 show index 命令查看创建的索引，可以看到索引类型是 FULLTEXT，表示全文索引。

```
mysql> show index from IT_desc \G;     // 具体有关查看内容见下一小节讲解
*************************** 1. row ***************************
       Table: IT_desc
  Non_unique: 1
```

```
    Key_name: descript
 Seq_in_index: 1
  Column_name: descript
    Collation: NULL
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: FULLTEXT
      Comment:
Index_comment:
1 row in set (0.00 sec)
```

（5）多列索引只需要在创建索引时指定多列即可。

```
mysql> create index salary_age_index on IT_salary( 薪资 , 年龄 );
```

使用 show index 命令查看创建的索引。

```
mysql> show index from IT_salary;
| Table      |  Non_unique | Key_name          | Seq_in_index | Column_name
| IT_salary |         1 | salary_age_index  |          1 | 薪资
| IT_salary |         1 | salary_age_index  |          2 | 年龄
```

因为是使用两个列“薪资”和“年龄”创建的索引，salary_age_index 名称的索引显示有两个，分别对应这两个列。

（6）不需要使用索引时，应删除索引，命令格式如下：

```
DROP INDEX index_name ON talbe_name;
ALTER TABLE table_name DROP INDEX index_name;
ALTER TABLE table_name DROP PRIMARY KEY;
```

第一条命令是直接删除索引，第二条是修改表时删除索引，第三条是删除主键索引。下面演示直接删除索引的方式。

```
mysql> drop index salary_age_index on IT_salary;
```

使用 show index 命令查看删除后索引是否存在。

```
mysql> show index from IT_salary;
| Table      | Non_unique | Key_name          | Seq_in_index | Column_name |
| IT_salary |         0 | PRIMARY           |          1 | 员工 ID     |
| IT_salary |         0 | salary_unique_in dex |          1 | 姓名        |
| IT_salary |         1 | salary_index      |          1 | 薪资        |
```

名称是 salary_age_index 的索引已经被删除。

（7）修改表结构，删除某一列，如果此列有索引存在，相应的索引也会被删除。

```
mysql> ALTER TABLE IT_salary DROP COLUMN 薪资 ;
Query OK, 2 rows affected (0.09 sec)
```

```
Records: 2  Duplicates: 0  Warnings: 0
mysql> show index from IT_salary;
| Table      | Non_unique | Key_name          | Seq_in_index | Column_name |
| IT_salary |          0 | PRIMARY           |            1 | 员工 ID     |
| IT_salary |          0 | salary_unique_index |          1 | 姓名        |
2 rows in set (0.00 sec)
```

很明显，删除“薪资”列后，salary_index 索引也被删除了。当索引是多列时，前面已经讲到是针对每个列都创建了索引，并且索引名称是相同的，此时若删除某个列，只是删除它自己对应的索引，而其他列的索引依然存在。

```
mysql> create index salary_age_index on IT_salary( 薪资 , 年龄 );
mysql> show index from IT_salary;
| Table      | Non_unique | Key_name          | Seq_in_index | Column_name
| IT_salary |          1 | salary_age_index  |            1 | 薪资
| IT_salary |          1 | salary_age_index  |            2 | 年龄
mysql> ALTER TABLE IT_salary DROP COLUMN 薪资 ;
mysql> show index from IT_salary;
| Table    | Non_unique | Key_name          | Seq_in_index | Column_name
| IT_salary |          1 | salary_age_index  |            1 | 年龄
```

salary_age_index 索引是包括“薪资”“年龄”的多列索引，当删除列“薪资”后，salary_age_index 对应“薪资”的索引也被删除了，但对应“年龄”的索引依然存在。

2. 索引查看方法

查看索引的命令前面已经使用了，这里再详细介绍一下，有两种命令可以查看索引，它们的显示结果完全相同。命令格式如下：

```
SHOW  INDEX  FROM  tablename;
```

或

```
SHOW KEYS FROM tablename;
```

在命令后面加上“\G”，可以竖向显示索引的信息。

```
mysql> show keys from IT_salary \G;
*************************** 1. row ***************************
        Table: IT_salary
   Non_unique: 0
     Key_name: PRIMARY
  Seq_in_index: 1
  Column_name: 员工 ID
    Collation: A
  Cardinality: 5
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
```

```
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: IT_salary
   Non_unique: 0
     Key_name: salary_unique_index
  Seq_in_index: 1
  Column_name: 姓名
    Collation: A
  Cardinality: 5
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 3. row ***************************
        Table: IT_salary
   Non_unique: 1
     Key_name: salary_index
  Seq_in_index: 1
  Column_name: 薪资
    Collation: A
  Cardinality: 5
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
3 rows in set (0.00 sec)
```

索引的显示参数描述如表 2-1 所示。

表 2-1　索引详细信息

显示参数	描述
Table	表的名称
Non_unique	索引值的唯一性，0 表示唯一性，1 表示非唯一性
Key_name	索引的名称
Seq_in_index	索引中的列序列号，从 1 开始
Column_name	列名称

2.2 MySQL 事务

数据库事务（Database Transaction）是指作为单个逻辑工作单元执行的一系列操作，

要么完整地执行，要么完全地不执行。正常情况下，这些操作将顺利进行，最终操作成功。但是，如果在这一系列过程中任何一个环节出了差错，数据库中所有信息都必须保持第一步操作前的状态不变。否则，数据库的信息将会一片混乱而不可预测。

2.2.1　事务的概念及特点

1. 事务的概念

事务是一种机制、一个操作序列，包含了一组数据库操作命令，并且把所有的命令作为一个整体一起向系统提交或撤销操作请求，即这一组数据库命令要么都执行，要么都不执行。事务是一个不可分割的工作逻辑单元，在数据库系统上执行并发操作时，事务是最小的控制单元。事务适用于用户同时操作数据库系统的场景，如银行、保险公司及证券交易系统等，通过事务的整体性以保证数据的一致性。事务是保证了一组操作的平稳性和可预测性的技术。

2. 事务的 ACID 特性

事务具体有四个特性：原子性（automicity）、一致性（consistency）、隔离性（isolation）、持久性（durability）。

（1）原子性：事务是一个完整的操作，各元素是不可分的，即原子的。事务中的所有元素必须作为一个整体提交或回滚。如果事务中的任何元素失败，则整个事务将失败。

（2）一致性：当事务完成时，数据必须处于一致状态；在事务开始之前，数据库中存储的数据处于一致状态；在正在进行的事务中，数据可能处于不一致的状态；当事务成功完成时，数据必须再次回到已知的一致状态。

（3）隔离性：对数据进行修改的所有并发事务是彼此隔离的，这表明事务必须是独立的，它不应以任何方式依赖于或影响其他事务。修改数据的事务可以在另一个使用相同数据的事务开始之前访问这些数据，或者在另一个使用相同数据的事务结束之后访问这些数据。

（4）持久性：指不管系统是否发生了故障，事务处理的结果都是永久的。一旦事务被提交，事务的效果会被永久地保留在数据库中。

2.2.2　事务的操作

1. MySQL 操作事务

默认情况下 MySQL 的事务是自动提交的。之前我们用 SQL 操作数据库时，一条语句执行后，系统会自动执行事务提交。当需要把一组语句作为一个事务提交时，需要手动对事务进行控制。手动控制事务有两种方法，一种是使用事务处理命令控制，另一种是使用 set 设置事务的处理方式。下面介绍这两种方法。

2. 使用事务命令控制事务

MySQL 中使用命令控制事务需要用到 3 个命令：

- begin：表示开始一个事务，后面会有多条数据库操作语句执行。
- commit：表示提交一个事务，对应前面的 begin 操作，它们之间的数据库操作语句一起完成。
- rollback：表示回滚一个事务，在 begin 和 commit 之间，如果某一个数据库操作语句出现错误，执行 rollback 回滚，数据库回到 begin 之前的状态，也就是操作语句都没执行。

（1）下面演示事务命令的使用方式，假设需要往数据表 IT_salary 中插入两条数据，它们在一个事务中提交。

```
mysql> begin;
mysql> insert into IT_salary values('nginx 工程师 ',' 小王 ','38','016',' 大专 ','10','12000');
mysql> insert into IT_salary values('apache 工程师 ',' 小李 ','29','019',' 大专 ','9','15500');
mysql> commit;
mysql> select * from IT_salary;
+----------------------+----------+--------+----------+--------+--------+--------+
| 岗位类别             | 姓名     | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------------+----------+--------+----------+--------+--------+--------+
| nginx 工程师         | 小王     |   38 |     16 | 大专 |    10 | 12000 |
| apache 工程师        | 小李     |   29 |     19 | 大专 |     9 | 15500 |
+----------------------+----------+--------+----------+--------+--------+--------+
```

使用 begin 开始事务，然后执行了两条插入语句，最后用 commit 提交事务，此时两条数据插入到了数据表中，这两条语句是作为一个整体操作的。

（2）上面的例子演示了事务的基本操作方式，那么如果使用 begin 开始事务，但不使用 commit 提交会出现什么情况。

```
mysql> delete from IT_salary;
mysql> begin;
mysql> insert into IT_salary values('nginx 工程师 ',' 小王 ','38','016',' 大专 ','10','12000');
mysql> insert into IT_salary values('apache 工程师 ',' 小李 ','29','019',' 大专 ','9','15500');
mysql> select * from IT_salary;
+----------------+--------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名   | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+--------+--------+----------+--------+--------+--------+
| nginx 工程师   | 小王   |   38 |     16 | 大专 |    10 | 12000 |
| apache 工程师  | 小李   |   29 |     19 | 大专 |     9 | 15500 |
+----------------+--------+--------+----------+--------+--------+--------+
```

可以看到依然能查询出插入的数据，但这只是在当前开启的事务中可以看到，因为没有提交，它们并不是真正插入到了数据库中。现在退出 MySQL 的连接，重新进入 MySQL 再进行查询。

```
mysql> exit;
Bye
[root@www ~]# mysql -uroot -p
……    // 省略部分信息
mysql> use imployee_salary;
Database changed
mysql> select * from IT_salary;
Empty set (0.00 sec)
```

此时数据表中并没有之前插入的数据，说明使用 begin 开始事务，执行操作语句后，必须使用 commit 进行提交，否则数据是不能自动提交的，也就是多条数据操作语句作为一个整体需要使用 commit 进行整体的事务提交。

（3）rollback 回滚的使用，假设插入一条数据后，需要恢复到插入前数据库的状态，使用 rollback 执行事务回滚。

```
mysql> delete from IT_salary;                    // 删除所有数据
mysql> begin;                                    // 开始事务
mysql> insert into IT_salary values('nginx 工程师 ',' 小王 ','38','016',' 大专 ','10','12000');
mysql> select * from IT_salary;
+---------------+--------+--------+----------+--------+--------+--------+
| 岗位类别      | 姓名   | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+---------------+--------+--------+----------+--------+--------+--------+
| nginx 工程师  | 小王   |   38   |    16    | 大专   |   10   | 12000  |
+---------------+--------+--------+----------+--------+--------+--------+
1 row in set (0.00 sec)
mysql> rollback;
mysql> select * from IT_salary;
Empty set (0.01 sec)
+----------------------+-----------+--------+----------+--------+--------+
```

可以看到，使用 begin 开始事务，然后执行了插入命令，查询表数据可以看到插入的数据，但是执行 rollback 回滚命令后，再查看表数据，先前插入的数据已经没有了，rollback 使数据表回滚到 begin 开始之前的状态。

（4）很多时候一个事务会包含多条语句，而出现问题需要回滚时，并不一定是要回滚到 begin 之前的状态，有可能是某条语句执行后的状态，这时要使用 savepoint 定义回滚点，rollback 决定回滚到的位置。

```
mysql> delete from IT_salary;    // 删除所有数据
mysql> begin;                    // 开始事务
mysql> insert into IT_salary values('nginx 工程师 ',' 小王 ','38','016',' 大专 ','10','12000');
mysql> savepoint s1;             // 定义回滚点
mysql> insert into IT_salary values('apache 工程师 ',' 小李 ','29','019',' 大专 ','9','15500');
mysql> savepoint s2;
```

执行第一条插入语句后，定义了回滚点 s1，执行第二条插入语句后，定义了回滚点 s2，如果后面直接使用 rollback 命令，这两条插入语句都将失效，现在使用回滚点

进行回滚。

```
mysql> rollback to savepoint s1;
mysql> select * from IT_salary;
+----------------+--------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名   | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+--------+--------+----------+--------+--------+--------+
| nginx 工程师   | 小王   |   38   |    16    | 大专   |   10   | 12000  |
+----------------+--------+--------+----------+--------+--------+--------+
1 row in set (0.00 sec)
```

回滚到了 s1，第一条插入的数据可以查询出来，但是并没有提交，如果需要保存到数据库，使用 commit 命令提交。

3. 使用 set 设置控制事务

前面已经提到 MySQL 默认是自动提交事务，也可以修改为不自动提交，使用 set 命令操作：

```
set autocommit = 0 : 禁止自动提交
set autocommit = 1 : 开启自动提交
```

（1）实际上 MySQL 启动时 autocommit 的值默认是 1，修改为 0，则需要手动提交，也就是使用前面学到的 commit 命令，回滚使用 rollback 命令。

```
mysql> set autocommit=0;
mysql> insert into IT_salary values('nginx 工程师',' 小王 ','38','016',' 大专 ','10','12000');
mysql> insert into IT_salary values('apache 工程师',' 小李 ','29','019',' 大专 ','9','15500');
mysql> commit;
mysql> select * from IT_salary;
+----------------------+-----------+--------+----------+--------+--------+--------+
| 岗位类别             | 姓名      | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------------+-----------+--------+----------+--------+--------+--------+
| nginx 工程师         | 小王      |   38   |    16    | 大专   |   10   | 12000  |
| apache 工程师        | 小李      |   29   |    19    | 大专   |    9   | 15500  |
+----------------------+-----------+--------+----------+--------+--------+--------+
```

使用 set autocommit=0 设置禁止自动提交，最后使用 commit 提交事务。

（2）set autocommit=0 设置禁止自动提交，不使用 commit 提交操作语句，和使用 begin 的情况相同，执行的语句都会失效。

```
mysql> set autocommit=0;
mysql> insert into IT_salary values('nginx 工程师',' 小王 ','38','016',' 大专 ','10','12000');
mysql> insert into IT_salary values('apache 工程师',' 小李 ','29','019',' 大专 ','9','15500');
mysql> select * from IT_salary;
+----------------+--------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名   | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+--------+--------+----------+--------+--------+--------+
| nginx 工程师   | 小王   |   38   |    16    | 大专   |   10   | 12000  |
```

```
| apache 工程师  | 小李 |   29 |    19 | 大专 |    9 | 15500 |
+----------------+--------+--------+----------+--------+--------+--------+
mysql> exit;
Bye
[root@www ~]# mysql -uroot -p
Enter password:
mysql> use imployee_salary;
Database changed
mysql> select * from IT_salary;
Empty set (0.00 sec)
```

可以看到如果不使用 commit 提交事务，退出 MySQL 重新进入后，插入的数据不存在。

本章总结

- 数据库索引分为普通索引、唯一性索引、主键索引、全文索引、多列索引。
- 数据库索引可以协助快速查询表中数据，但并不是任何字段都需要创建索引。
- 数据库事务的 ACID 特性。
- MySQL 事务命令有 begin、rollback、commit、savepoint。

本章作业

1．简述数据库索引的种类。

2．简述数据库事务的 ACID 特性。

3．使用 begin、savepoint、rollback 实现回滚点的事务操作，首先使用 begin 开始事务，然后插入数据 1，创建回滚点 1，再插入数据 2，创建回滚点 2，分别回滚到回滚点 1 和回滚点 2，查看数据的当前状态。

4．用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



随手笔记



第3章

MySQL 数据库管理

技能目标

- 掌握 MySQL 用户管理
- 掌握 MySQL 日志管理
- 了解 MySQL 数据乱码的解决方法

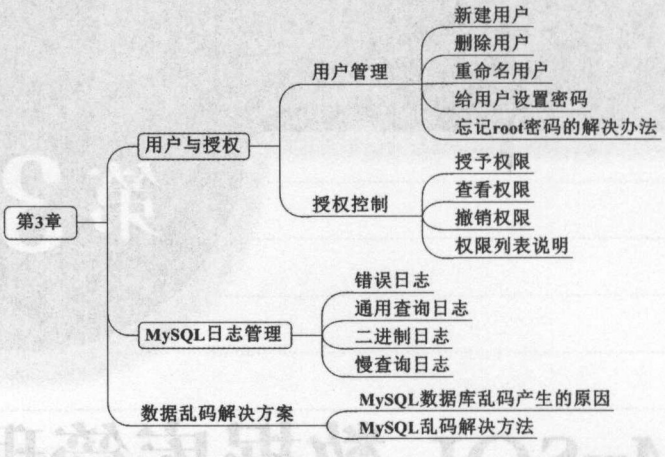
本章导读

在企业信息化的过程中，数据库中库和表都会大量存在，需要分配给管理者合适的权限进行管理操作。合理地分配权限，可以使数据库管理井然有序，各个管理者只需要关注自己负责的内容，也可避免误操作对系统造成损失。

任何数据库系统都有日志文件，用来记录操作和错误等信息，有助于管理、排查错误，保证数据库正常运行。

知识服务





3.1 用户与授权

数据库是信息系统中非常重要的一个环节，合理高效地对它进行管理是很重要的工作。通常是由总管理员创建不同的管理账户，然后分配不同的操作权限，把这些账户交给相应的管理人员使用。

3.1.1 用户管理

1. 新建用户

新建用户的命令格式如下：

```
CREATE USER 'username'@'host' [IDENTIFIED BY [PASSWORD] 'password'];
```

其中：

- username：将创建的用户名。
- host：指定该用户在哪些主机上可以登录，可使用IP地址、网段、主机名的形式，如果是本地用户可用 localhost，如果想让该用户可以从任意远程主机登录，可以使用通配符 %。
- password：因为 MySQL 5.7 版本启用了密码增强插件，新密码必须符合密码复杂性要求，用户的登录密码不能为空。密码通常是需要经过加密处理再写入数据库，在不使用 PASSWORD 关键字时，可以使用明文密码，插入到数据库时由 MySQL 自动加密。如果使用 PASSWORD 关键字，可以直接插入密文，MySQL 不再进行加密处理，可以使用 MySQL 的 select password('passowd') 语句获取加密后的密文，添入密文做为密码。下面例子是不使用 PASSWORD 关键字，直接使用明文密码。

```
[root@www ~]# mysql -uroot -p
```

```
mysql>  CREATE USER 'user1'@'localhost' IDENTIFIED BY 'H@o123.com';
Query OK, 0 rows affected (0.00 sec)          |
```

下面是使用 PASSWORD 关键字，使用密文作为密码。

```
mysql> select password('H@123.com');
+-------------------------------------------+
| password('H@123.com')                     |
+-------------------------------------------+
| *18C0DC50249A1DDE76D964231B85E76F288713B4 | // 密码 'H@o123.com' 加密后的密文
+-------------------------------------------+
1 row in set (0.05 sec)

mysql> CREATE USER 'user2'@'localhost' IDENTIFIED BY PASSWORD
          '*18C0DC50249A1DDE76D964231B85E76F288713B4';
Query OK, 0 rows affected (0.00 sec)
```

创建后的用户是保存在 mysql 数据库的 user 表里，使用查询语句可以查看创建的用户。

```
mysql> use mysql;
Database changed

mysql> select User,authentication_string,Host from user;
+-----------+-------------------------------------------+-----------+
| User      | authentication_string                     | Host      |
+-----------+-------------------------------------------+-----------+
| root      | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost |
| user1     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| user2     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
+-----------+-------------------------------------------+-----------+
4 rows in set (0.00 sec)
```

用户 user1、user2 的 password 字段值相同，原始密码都是“H@123.com”，只是采用不同的明文和密文的形式创建的，最后数据库中是一致的。

下面用新创建的用户登录数据库，输入密码后可以正常登录，说明创建用户成功。

```
[root@www ~]# mysql -uuser1 -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.17 MySQL Community Server (GPL)
```

2. 删除用户

删除用户的命令格式如下：

```
DROP  USER  'username' @'host';
mysql> select User,authentication_string,Host from user;
```

```
+-----------+-------------------------------------------+-----------+
| User      | authentication_string                     | Host      |
+-----------+-------------------------------------------+-----------+
| root      | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost |
| user1     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| user2     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
+-----------+-------------------------------------------+-----------+
4 rows in set (0.00 sec)
mysql> drop user 'user2'@'localhost';
Query OK, 0 rows affected (0.00 sec)
mysql> select User,authentication_string,Host from user;
+-----------+-------------------------------------------+-----------+
| User      | authentication_string                     | Host      |
+-----------+-------------------------------------------+-----------+
| root      | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost |
| user1     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
+-----------+-------------------------------------------+-----------+
3 rows in set (0.00 sec)
```


删除 'user2'@'localhost' 这个用户后，数据库中已经没有该用户了。

既然用户是存储在 user 表中，直接使用 DELETE 语句也可以对它执行删除，但和 DROP 是有区别的。DROP 不只删除用户表，还会把相关的权限删除，而 DELETE 只删除用户，权限在数据库中依然存在。

3. 重命名用户

重命名用户的命令格式如下：

```
RENAME USER 'old_user'@ ' host'  TO  'new_user' @ 'host' ;
```

old_user 是旧的用户名，new_user 是新的用户名。重新加入用户 user2 后，把它改名为 user3。

```
mysql>  CREATE USER 'user2'@'localhost' IDENTIFIED BY 'H@o123.com';
mysql> select User,authentication_string,Host from user;
+-----------+-------------------------------------------+-----------+
| User      | authentication_string                     | Host      |
+-----------+-------------------------------------------+-----------+
| root      | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost |
| user1     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
| user2     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost |
+-----------+-------------------------------------------+-----------+
4 rows in set (0.00 sec)
mysql> rename user 'user2'@'localhost' to 'user3'@'192.168.1.101';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select User,authentication_string,Host from user;
+-----------+-------------------------------------------+---------------+
| User      | authentication_string                     | Host          |
+-----------+-------------------------------------------+---------------+
| root      | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost     |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost     |
| user1     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost     |
| user3     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | 192.168.1.101 |
+-----------+-------------------------------------------+---------------+
4 rows in set (0.00 sec)
```

用户 user2 已经改为 user3，主机也由 host 改为了 192.168.1.101。

4. 给用户设置密码

修改用户密码的方式有两种，可以修改当前登录用户的密码或修改其他用户的密码。

（1）修改当前登录用户密码的命令格式如下：

```
SET  PASSWORD = PASSWORD('password');
```

使用函数 password() 对密码加密，退出后重新登录，需要使用新密码。

```
mysql> set password =password('h@O123.com');
Query OK, 0 rows affected (0.00 sec)

mysql> quit;
Bye
[root@www ~]# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

（2）修改其他用户密码的命令格式如下：

```
SET  PASSWORD = 'username' @ 'host' = PASSWORD('password') ;
```

同样需要使用 password() 对密码加密。下面例子是修改其他用户密码。

```
mysql> set password for 'user1'@'localhost' = password('h@O123.com');
Query OK, 0 rows affected (0.01 sec)
mysql> quit;
Bye
[root@www ~]# mysql -uuser1 -ph@O123.com
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

修改后用其他用户登录，需要使用新的密码。

5. 忘记 root 密码的解决办法

使用 MySQL 时，如果忘记了其他用户的密码，可以使用 root 用户重新设置，但是如果忘记了 root 的密码，就需要采用特殊的方法进行操作。

直接修改授权表可以修改 root 的密码，下面介绍它的使用步骤。

（1）停止 mysqld 服务进程。

```
[root@www ~]# systemctl stop mysqld
[root@www ~]# netstat -utpln |grep 3306    // 检测 mysql 是否已经停止
```

（2）使用 mysqld_safe 结合 skip-grant-tables 启动数据库，它的作用是用户登录时不使用授权表，所以用户可以不使用密码直接登录。

```
[root@www ~]# mysqld_safe --skip-grant-tables&
[root@www ~]# netstat -utpln |grep 3306  // 检测 mysql 是否已经启动
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      24386/mysqld
```

（3）可以不使用密码直接登录到 mysql，使用 update 修改 root 的密码。

```
[root@www ~]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> update mysql.user set authentication_string=password('123.Com') where user='root';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1
```

（4）刷新数据库，使用 update 修改密码后，只是在数据库中进行了修改，内存中的数据并没有修改。flush privileges 的作用就是把当前 user 和 privilege 表中的用户信息和权限设置从 mysql 库（MySQL 数据库的内置库）提取到内存里。MySQL 用户数据和权限修改后，希望在“不重启 MySQL 服务”的情况下直接生效，就需要执行这个命令。

```
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

（5）使用新密码做登录测试，登录成功说明修改成功。

```
[root@www ~]# mysql -uroot -p123.Com
Welcome to the MySQL monitor. Commands end with ; or \g.
```

3.1.2 授权控制

在 MySQL 中，权限设置非常重要，分配权限可以清晰地划分责任，管理人员只需要关注自己的任务即可，最重要的是保证系统数据的安全。

1. 授予权限

（1）权限控制主要是出于安全因素，因此需要遵循以下几个经验原则：

1）只授予能满足需要的最小权限，防止用户误操作和干坏事。比如用户只是需要查询，只需赋予 select 权限就可以了，不要给用户赋予 update、insert 或者 delete 权限。

2）创建用户的时候限制用户的登录主机，一般是限制成指定 IP 或者内网 IP 段。

3）初始化数据库时删除没有密码的用户。安装完数据库时会自动创建一些用户，这些用户默认没有密码。

4）为每个用户设置满足密码复杂度的密码。

5）定期清理不需要的用户。回收权限或者删除用户。

（2）授予权限使用 GRANT 命令，命令格式如下：

```
GRANT 权限列表 ON 库名 . 表名 TO 用户名 @ 主机地址 [IDENTIFIED BY ' 密码 ']。
```

命令格式很明确，是指定用户允许它操作某些表，对这些表拥有相应的操作权限。下面演示 GRANT 的使用方式。

```
[root@www ~]# mysql -uroot –ph@O123.com
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> grant select on imployee_salary.IT_salary to 'user1'@localhost identified by 'h@O123.com';
Query OK, 0 rows affected (0.00 sec)
```

使用户 user1 可以在主机 localhost 连接，密码是 h@O123.com，它拥有对数据库表 imployee_salary.IT_salary 的 select 权限。

使用 user1 登录 MySQL 进行验证。

```
[root@www ~]# mysql -uuser1 -ph@O123.com
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> select * from imployee_salary.IT_salary;
Empty set (0.12 sec)
mysql> insert into imployee_salary.IT_salary values('1','nginx 工 程师 ',' 小王 ','38','12000');
ERROR 1142 (42000): INSERT command denied to user 'user1'@'localhost' for table 'IT_salary'
mysql>
```

select 语句可以正常执行，但是执行 insert 语句显示没有足够的权限。

使用 GRANT 时有些问题需要注意：

1）当用户名和主机名在数据库中不存在时，用户和主机名被创建，也就是 user 表中多了一个用户数据，和使用创建新用户命令效果是相同的，登录密码是后面指定的密码。

```
[root@www ~]# mysql -uroot -ph@O123.com
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> grant select on imployee_salary.IT_salary to 'user5'@localhost identified by 'H@o123.com';
mysql> select User,authentication_string,Host from mysql.user;
+-----------+-------------------------------------------+---------------+
| User      | authentication_string                     | Host          |
+-----------+-------------------------------------------+---------------+
| root      | *F649DE00891072716E261E7BA50A80D5D8854BE1 | localhost     |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost     |
| user1     | *F649DE00891072716E261E7BA50A80D5D8854BE1 | localhost     |
| user3     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | 192.168.1.101 |
| user5     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | localhost     |
+-----------+-------------------------------------------+---------------+
5 rows in set (0.01 sec)
```

用户 user5 不存在，使用 GRANT 语句后，创建了用户 user5。

2）当用户名和主机名在数据库中已经存在，后面设置的新密码可以覆盖旧密码，相当于修改密码的功能。

```
mysql> grant select on imployee_salary.IT_salary to 'user5' @localhost identified by 'h@O123.com';
Query OK, 0 rows affected (0.00 sec)
mysql> select User,authentication_string,Host from mysql.user;
+-----------+-------------------------------------------+---------------+
| User      | authentication_string                     | Host          |
+-----------+-------------------------------------------+---------------+
| root      | *F649DE00891072716E261E7BA50A80D5D8854BE1 | localhost     |
| mysql.sys | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | localhost     |
| user1     | *F649DE00891072716E261E7BA50A80D5D8854BE1 | localhost     |
| user3     | *18C0DC50249A1DDE76D964231B85E76F288713B4 | 192.168.1.101 |
| user5     | *F649DE00891072716E261E7BA50A80D5D8854BE1 | localhost     |
+-----------+-------------------------------------------+---------------+
5 rows in set (0.00 sec)
```

用户 user5 在数据库中已经存在，使用 GRANT 语句后，因为后面指定的新密码与旧密码不同，新密码覆盖了旧密码。

2. 查看权限

查看用户拥有的权限可以使用 SHOW GRANTS 命令。命令格式如下：

```
SHOW GRANTS FOR 'username'@' 主机地址 ';
```

下面查看用户 user5 的权限。

```
mysql> show grants for 'user5'@'localhost';
+-----------------------------------------------------------------------+
| grants for user5@localhost                                            |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'user5'@'localhost'                             |
| GRANT SELECT ON 'imployee_salary'.'IT_salary' TO 'user5'@'localhost' |
+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

显示 user5 对表 imployee_salary.IT_salary 拥有 select 权限，与之前设置的相同。

3. 撤销权限

使用 REVOKE 语句可以撤销指定用户的数据库权限。命令格式如下：

```
REVODE 权限列表 ON 数据库名 . 表名 FROM 用户 @ 主机地址。
```

首先对 user5 授予了 select 和 update 权限，使用 revoke 撤销 update 权限后，user5 只拥有 select 权限，说明操作是成功的。

```
mysql> grant select,update on imployee_salary.IT_salary to 'user5'@localhost identified by 'h@O123.com';
Query OK, 0 rows affected (0.00 sec)
mysql> show grants for 'user5'@'localhost';
```

```
+-----------------------------------------------------------------+
|grants for user5@localhost                                       |
+-----------------------------------------------------------------+
|GRANT USAGE ON *.* TO 'user5'@'localhost'                        |
|GRANT SELECT, UPDATE ON 'imployee_salary'.'IT_salary' TO 'user5'@'localhost' |
+-----------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> revoke update on imployee_salary.IT_salary from 'user5'@'localhost';
Query OK, 0 rows affected (0.00 sec)
mysql> show grants for 'user5'@'localhost';
+-----------------------------------------------------------------+
| grants for user5@localhost                                      |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'user5'@'localhost'                       |
| GRANT SELECT ON 'imployee_salary'.'IT_salary' TO 'user5'@'localhost' |
+-----------------------------------------------------------------+
2 rows in set (0.00 sec)
```

如需要撤销用户的所有权限可以使用 all 关键字，下面演示撤销用户 user5 的所有权限。

```
mysql> revoke all on imployee_salary.IT_salary from 'user5'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

4. 权限列表说明

操作权限有很多种，如表 3-1 所示，可以精确地控制用户权限。

表 3-1　权限列表

权限	说明
ALL	设置 GRANT OPTION 之外的所有权限
ALTER	允许使用 ALTER TABLE
CREATE	允许使用 CREATE TABLE
CREATE USER	允许使用 CREATE USER
DELETE	允许使用 DELETE
INDEX	允许使用 INDEX
INSERT	允许使用 INSERT
SELECT	允许使用 SELECT
UPDATE	允许使用 UPDATE
DROP	允许使用 DROP TABLE
REPLICATION SLAVE	允许从主服务器中读取二进制日志文件
SHOW DATABASES	允许显示所有数据库

3.2 MySQL 日志管理

由于日志文件是掌握数据库运行状态的重要参考，因此日志文件的维护也有十分重要的意义。

MySQL 的日志类型有错误日志、通用查询日志、二进制日志、慢查询日志。

1. 错误日志

（1）错误日志主要记录当 MySQL 启动和停止时，以及在运行过程中发生任何错误时的相关信息。错误日志默认保存在 MySQL 的安装路径 data 文件夹下，后缀名是 .err。

```
[root@www ~]# cd /usr/local/mysql/data/
[root@www data]# ls
www.err // 省略内容
```

（2）在 MySQL 的配置文件中，可以指定日志文件的保存位置和日志的文件名。log-error=file_name 选项来指定保存错误日志的位置，file_name 指定日志文件名，如果没有指定文件名，使用 host_name.err 作为文件名。

```
[root@www data]# vi /etc/my.cnf
// 省略内容
log-error=/usr/local/mysql/data/mysql_error.log
// 省略内容
```

重启 MySQL 后，生成 mysql_error.log 用来记录错误日志。

```
[root@www data]# systemctl restart mysqld
 [root@www data]# ls
mysql_error.log
// 省略内容
```

需要注意的是，MySQL 必须对指定的文件夹有读写的权限，否则无法操作文件。

2. 通用查询日志

（1）通用查询日志用来记录 MySQL 的所有连接和语句，默认是关闭的。使用 SHOW 语句可以查询出日志的信息。

```
mysql> show variables like 'general%';
+------------------+-------------------------------+
| Variable_name    | Value                         |
+------------------+-------------------------------+
| general_log      | OFF                           |
| general_log_file | /usr/local/mysql/data/www.log |
+------------------+-------------------------------+
2 rows in set (0.01 sec)
```

general_log 对应的是通用查询日志，它的值是 OFF，表示是关闭的。

（2）修改 MySQL 配置文件的 general_log=ON 选项，可以打开通用查询日志，general_log_file=file_name 定义通用查询日志的位置，如果没有指定 file_name 的值，默认名是 host_name.log。

```
[root@www data]# vi /etc/my.cnf
general_log=ON
general_log_file=/usr/local/mysql/data/mysql_general.log
```

（3）重启 MySQL 后，通用查询日志生效。

```
[root@www data]# systemctl restart mysqld
 [root@www data]# ls
mysql_general.log
// 省略内容
```

（4）使用 SHOW 语句查看通用查询日志已经打开，general_log 对应的值是 ON。

```
[root@www data]# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> show variables like 'general%';
+------------------+---------------------------------------+
| Variable_name    | Value                                 |
+------------------+---------------------------------------+
| general_log      | ON                                    |
| general_log_file | /usr/local/mysql/data/mysql_general.log |
+------------------+---------------------------------------+
2 rows in set (0.01 sec)
```

3. 二进制日志

（1）二进制日志用来记录所有更新了数据或者已经潜在更新了数据的语句，记录了数据的更改，主要目的是在恢复数据时能够最大程度地恢复数据库。二进制日志默认是开启的，在 data 文件夹下，以 mysql-bin 命名，数据量大时，它会自动分割成多个日志文件，以数字做为扩展名。

```
[root@www data]# ls
mysql-bin.000004  mysql-bin.000013  mysql-bin.000022  mysql-bin.000031   mysql-bin.000005
    mysql-bin.000014  mysql-bin.000023  mysql-bin.000032
mysql-bin.000006  mysql-bin.000015  mysql-bin.000024  mysql-bin.000033   mysql-bin.000007
    mysql-bin.000016  mysql-bin.000025  mysql-bin.000034   mysql-bin.000008  mysql-bin.000017
    mysql-bin.000026  mysql-bin.000035   mysql-bin.000009   mysql-bin.000018  mysql-bin.000027
    mysql-bin.000036
```

（2）二进制日志可以使用 MySQL 的工具 mysqlbinlog 查看二进制文件。

```
 [root@www data]# mysqlbinlog mysql-bin.000001
// 省略内容
```

```
# at 1027593
#160528 21:08:39 server id 1  end_log_pos 1027663 Query thread_id=1 exec_time=0 error_code=0
SET TIMESTAMP=1464440919/*!*/;
COMMIT
/*!*/;
// 省略内容
```

在本机里有一条事务操作 commit，操作的时间是 16 年 5 月 28 日 21 点 8 分 39 秒，这就是二进制文件的内容格式，MySQL 根据这样的信息做恢复数据库操作。

4. 慢查询日志

慢查询日志记录所有执行时间超过 long_query_time 秒的 SQL 语句，用于找到哪些查询语句执行时间长，以便对其进行优化。默认慢查询日志是关闭的，slow_query_log 是慢查询的选项，默认是 OFF。

```
mysql>  show variables like '%slow%';     // 查看 MySQL 慢查询功能是否开启
+---------------------------+-----------------------------------+
| Variable_name             | Value                             |
+---------------------------+-----------------------------------+
| log_slow_admin_statements | OFF                               |
| log_slow_slave_statements | OFF                               |
| slow_launch_time          | 2                                 |
| slow_query_log            | OFF                               | // 状态为 OFF
| slow_query_log_file       | /usr/local/mysql/data/www-slow.log |
+---------------------------+-----------------------------------+
5 rows in set (0.01 sec)
```

（1）可以使用 SQL 语句开启慢查询功能。

```
mysql> set global slow_query_log=ON;   // 开启 MySQL 慢查询功能
Query OK, 0 rows affected (0.27 sec)

mysql> show variables like '%slow%';
+---------------------------+-----------------------------------+
| Variable_name             | Value                             |
+---------------------------+-----------------------------------+
| log_slow_admin_statements | OFF                               |
| log_slow_slave_statements | OFF                               |
| slow_launch_time          | 2                                 |
| slow_query_log            | ON                                | // 状态为 ON
| slow_query_log_file       | /usr/local/mysql/data/www-slow.log |
+---------------------------+-----------------------------------+
5 rows in set (0.01 sec)
```

慢查询时间设置默认为 10 秒，记录 10 秒内的查询，可以通过 global long_query_time 选项进行修改。

```
mysql> show variables like 'long_query_time';   // 查看慢查询时间设置
```

```
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)

mysql> set global long_query_time=5; // 更改慢查询时间设置
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'long_query_time';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 5.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)
```

使用 SHOW 语句查看慢查询日志已经打开，slow_query_log 对应的值是 ON。

```
[root@www data]# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql>  show variables like '%slow%';
+---------------------------+----------------------+
| Variable_name             | Value                |
+---------------------------+----------------------+
| log_slow_admin_statements | OFF                  |
| log_slow_slave_statements | OFF                  |
| slow_launch_time          | 2                    |
| slow_query_log            | ON                   |
| slow_query_log_file       |/usr/local/mysql/data/www-slow.log |
+---------------------------+----------------------+
5 rows in set (0.01 sec)
```

（2）也可以在 MySQL 的配置文件中进行设置，指定 slow_query_log = ON 选项启动慢查询功能，使用 slow_query_log_file =filename 指定存放慢查询日志位置，如果没有 filename 值，默认为 hotname-slow.log，如果没有给出绝对路径，文件保存在 data 目录下。long_query_time=1 设置了超过 1 秒执行的语句被记录。

```
[root@www data]# vi /etc/my.cnf
// 省略内容
slow_query_log=ON
slow_query_log_file=mysql_slow_query.log
long_query_time = 1
// 省略内容
```

重启 MySQL 后，慢查询日志 mysql_slow_query.log 被创建。经常检查慢查询日志，

当有查询语句被记录，就需要对其进行优化处理了。

```
[root@www data]#systemctl restart mysqld

[root@www data]# ls
mysql_slow_query.log
// 省略内容
```

（3）进行慢查询测试生成慢查询日志。

```
mysql> select sleep(6);    // 慢查询测试
+----------+
| sleep(6) |
+----------+
|        0 |
+----------+
1 row in set (6.00 sec)

mysql> show global status like '%slow%';    // 查看慢查询状态
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| Slow_launch_threads | 0     |
| Slow_queries        | 0     |
+---------------------+-------+
2 rows in set (0.01 sec)
```

查看生成的慢查询日志。

```
[root@www ~]# cat /usr/local/mysql/data/mysql_slow_query.log /usr/local/mysql/bin/mysqld, Version:
      5.7.17-log (Source distribution). started with:
Tcp port: 0  Unix socket: (null)
Time                 Id Command    Argument
```

3.3 数据乱码解决方案

1. MySQL 数据库乱码产生的原因

在计算机中任何数据都是以二进制存储的，要存储一个字符就要对它进行编码，用一个二进制数与其对应，这种对应的规则就是字符的编码。编码的规则有很多种，一种规则所编码的“字符”的集合就叫做“字符集”。在制定编码标准的时候，“字符的集合”和“编码”一般都是同时制定的，因此，平时我们所说的“字符集”，例如 GB 2312、GBK 和 UTF-8 等，除了有“字符的集合”这层含义外，同时也包含了“编码”的含义。

对中文处理可以使用 GB 2312 和 GBK，它是中文字符集，而 UTF-8 是对世界上

每个国家和地区的文字制定的字符集，并且 UTF-8 是可变存储长度的字符集，如英文字母只需要一个字节存储，节省了存储空间，所以数据库中通常采用 UTF-8 的字符集。

MySQL 服务器使用中经常会有乱码产生，主要有以下几种原因：

（1）服务器系统字符设置问题。

（2）数据表语系设置的问题。

（3）客户端连接语系的问题。

实际上对乱码问题原因的简单解释就是：在使用、存储数据的时候，各个环节设置的字符集如果不同，就会产生乱码。

2. MySQL 乱码解决方法

对于 MySQL 服务器，只要设置存储的字符集为 UTF-8，对应的客户端程序也使用相同编码，就不会产生乱码。MySQL 设置字符集的方式有多种：

（1）在创建数据库时指定字符集，使用参数 CHARACTER SET 设置字符集，COLLATE 是对字符集进行校对的规则，如下所示，创建的数据库字符集就是 UTF-8 了。

```
[root@www data]# mysql -uroot -ph@O123.com
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> create database chartest character set 'utf8' collate 'utf8_general_ci';
Query OK, 1 row affected (0.01 sec)
```

创建数据表时如果不指定字符集，默认按数据库的字符集指定，使用 show table status，可以查看表的字符集。

```
mysql> use chartest;
Database changed
mysql> create table idtest(id int(10));
Query OK, 1 row affected (0.01 sec)
mysql> show table status;
----+------------+-----------------+
| Name   | Engine | Version | Row_format  | Create_options |
| idtest | InnoDB |      10 | Dynamic     | utf8_general_ci |

1 row in set (0.00 sec)
```

创建表 idtest 时并没有指定符集，它默认使用数据库的字符集 UTF-8。

（2）创建表时也可以指定字符集，数据库中的表根据需要可以设置不同的字符集。

```
mysql> create table idtest2(id int(10)) default charset=gb2312;
Query OK, 0 rows affected (0.06 sec)
mysql> show table status;
| Name   | Engine | Collation
| idtest2 | InnoDB |gb2312_chinese_ci
| idtest  | InnoDB | utf8_general_ci |
----+------------+-----------------+
| Name   | Engine | Version | Row_format  | Create_options |
```

```
| idtest | InnoDB |      10 | Dynamic     | utf8_general_ci |
|idtest2 | InnoDB |      10 | Dynamic     | gb2312_chinese_ci|
```

表 idtest2 的字符集使用了 GB 2312，显示结果说明设置是正确的。

（3）使用 set names 也可以设置默认的字符集，但它是临时设置，MySQL 重启后失效。

```
mysql> set names utf8;
```

修改 MySQL 配置文件，加入 [client] 部分，以及添加选项 default-character-set 可以设置字符集，这时创建的数据库和表都不需要再指定，除非有其他字符集使用的地方再特别指定。通常使用的就是修改配置文件的方式，使数据库中所有存储内容都是 UTF-8 格式。

```
[root@www data]# vi /etc/my.cnf
// 省略内容
[client]
default-character-set=utf8
// 省略内容
```

只需要修改这一处，就可以统一数据库的字符集，重启后字符集生效。

本章总结

- MySQL 新建用户需要指定权限。
- 忘记 root 密码，可以跳过权限表进行修改。
- MySQL 日志包括错误日志、通用查询日志、二进制日志、慢速查询日志。
- MySQL 乱码问题可以采用指定字符集 UTF-8 的方式解决。

本章作业

1. 简述忘记 root 密码的修改过程。
2. 简述 MySQL 几种日志文件的作用。
3. 用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



扩展知识

1. 慢查询分析工具 mysqlsla

2. 使用 mysqlsla 分析慢查询日志

说明：请访问课工场网站。

第4章

MySQL 完全备份与恢复

技能目标

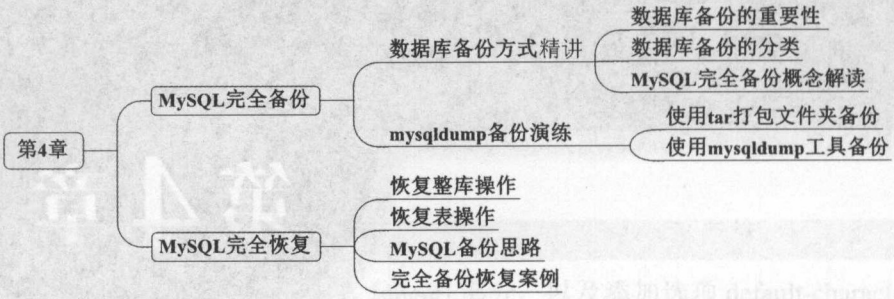
- 掌握使用 mysqldump 进行备份的方法
- 掌握 MySQL 数据恢复的方法

本章导读

随着办公自动化和电子商务的飞速发展，企业对信息系统的依赖性越来越高，数据库作为信息系统的核心担当着重要的角色。数据库备份，是在数据丢失的情况下，能及时恢复重要数据，防止数据丢失的一种重要手段。一个合理的数据库备份方案，应该能够在数据丢失时，有效地恢复数据，同时需要考虑技术实现难度和有效地利用资源。

知识服务





4.1 MySQL 完全备份

4.1.1 数据库备份方式精讲

1. 数据库备份的重要性

在生产环境中，数据的安全性是至关重要的，任何数据的丢失都可能产生严重的后果。数据库备份的重要性主要体现在：

（1）提高系统的高可用性和灾难可恢复性，在数据库系统崩溃时，没有数据库备份就没法找到数据。

（2）使用数据库备份还原数据库，是数据库崩溃时提供数据恢复最小代价的最优方案，如果让用户重新添加数据，代价未免太大。

（3）没有数据就没有一切，数据库备份是一种防范灾难的强力手段。

使用数据库的过程中，有多种原因造成数据的丢失：

（1）程序错误：指对数据库操作的程序难免有些错误，造成数据丢失。

（2）人为错误：指由于使用人员的误操作造成的数据被破坏，还有可能是黑客对系统攻击造成的数据丢失等。

（3）计算机失败：指运行数据库的服务器操作系统或软件损坏，有可能造成数据的损坏。

（4）磁盘失败：指硬盘等存储数据的硬件设备，长时间运行后可能损坏，造成数据丢失。

（5）灾难（如火灾、地震）和偷窃：指自然灾害等的发生，有可能造成数据丢失。

数据丢失会造成严重的经济损失，如携程网 2015 年 5 月 28 日数据库崩溃，每小时损失大概 106 万美元。所以在企业信息系统建设中，数据库的备份管理是非常重要的内容。

2. 数据库备份的分类

备份方式分很多种，从物理与逻辑的角度，备份分为以下几类：

（1）物理备份：指对数据库操作系统的物理文件（如数据文件、日志文件等）的备份。物理备份又可以分为脱机备份（冷备份）和联机备份（热备份）。

冷备份：在关闭数据库时进行的备份操作，能够较好地保证数据库的完整性。

热备份：在数据库运行状态中进行操作，这种备份方法依赖于数据库的日志文件。

（2）逻辑备份：指对数据库逻辑组件（如表等数据库对象）的备份。

从数据库的备份策略角度，备份可分为完全备份、差异备份和增量备份。

（1）完全备份：每次对数据进行完整的备份。可以备份整个数据库，包含用户表、系统表、索引、视图和存储过程等所有数据库对象。但它需要花费更多的时间和空间，所以，做一次完全备份的周期要长些。

（2）差异备份：备份那些自从上次完全备份之后被修改过的文件，只备份数据库部分的内容。它比最初的完全备份小，因为只包含自上次完全备份以来所改变的数据库。它的优点是存储和恢复速度快。

（3）增量备份：只有那些在上次完全备份或者增量备份后被修改的文件才会被备份。

表 4-1 所示为几种备份之间的区别。

表 4-1　备份方式比较

备份方式	完全备份	差异备份	增量备份
完全备份时的状态	表 1、表 2	表 1、表 2	表 1、表 2
第 1 次添加内容	创建表 3	创建表 3	创建表 3
备份内容	表 1、表 2、表 3	表 3	表 3
第 2 次添加内容	创建表 4	创建表 4	创建表 4
备份内容	表 1、表 2、表 3、表 4	表 3、表 4	表 4

完全备份每次都是把整个数据库中的内容进行备份。

差异备份在第 1 次添加内容后备份，是备份新添加的内容，第 2 次添加内容后备份，把第 1 次添加的内容也进行了备份，它是根据完全备份时的状态决定的。差异备份每次都是把完全备份之后所有的修改进行备份，不管前面是否做过差异备份。

增量备份只备份每次新修改的内容，第 1 次创建表 3，就对表 3 备份，第 2 次添加表 4 后，因为之前已经备份过表 3，所以只备份表 4 的内容即可。

关于这几种备份方式的动画演示，请登录课工场 APP 或官网 kgc.cn 观看。

3. MySQL 完全备份概念解读

MySQL 的备份方式主要有完全备份与增量备份。完全备份是对整个数据库的备份、数据库结构和文件结构的备份，保存的是备份完成时刻的数据库，是增量备份的基础。

完全备份的优点是备份与恢复操作简单方便，缺点是数据存在大量重复，占用大量的备份空间，备份的时间长。

在生产环境中，这两种备份方式都会使用，需要制定合理高效的方案达到备份数据的目的，避免数据丢失造成严重的后果。

4.1.2 mysqldump 备份演练

MySQL 数据库的备份可以采用两种方式，因为数据库实际上就是文件，直接打包数据库文件夹，或者是使用专用备份工具 mysqldump 都可以进行备份工作。

1. 使用 tar 打包文件夹备份

MySQL 的数据库文件默认都是保存在安装目录的 data 文件夹下面，可以直接保存 data 文件夹，但是占用的空间较大，可以使用 tar 打包压缩进行保存。

（1）数据库文件很大，可以使用压缩率较大的 xz 格式压缩，首先需要安装 xz 压缩格式工具。

```
[root@www mysql]# yum -y install xz
```

（2）对数据库文件夹 /usr/local/mysql/data/ 进行打包操作。

```
[root@www]#tar Jcvf /opt/mysql-$(date +%F).tar.xz /usr/local/mysql/data/
[root@www]# ls /opt/
mysql-2016-06-28.tar.xz
```

使用 ls 命令，查看打包命令是否运行成功，已经生成了备份文件。

（3）对比打包前后的文件大小，可以看到压缩的备份文件占用空间很小。

```
[root@www]# du -sh /opt/mysql-2016-06-28.tar.xz
592K        /opt/mysql-2016-06-28.tar.xz
[root@www]# du -sh /usr/local/mysql/data
36M         /usr/local/mysql/data
```

（4）如果数据库文件损坏数据丢失，可以解压缩备份文件，相当于做了数据恢复的工作。

```
[root@www]#tar Jxvf /opt/ mysql-2016-06-28.tar.xz /usr/local/mysql/data/
```

2. 使用 mysqldump 工具备份

前面介绍的对 MySQL 整个数据库目录压缩的方式，是备份数据库中所有的内容。使用 mysqldump 可以更加灵活地控制备份的内容，比如某几个表或库都可以单独备份。

（1）使用 mysqldump 命令对某些表进行完全备份，命令格式如下：

```
mysqldump -u 用户名 -p [ 密码 ] [ 选项 ] [ 数据库名 ] [ 数据表名 ]> / 备份路径 / 备份文件名
```

下面演示了对库 imployee_salary 中的表 IT_salary、IT_desc 进行备份，备份文件是 /opt/salary.sql。

```
[root@www mysql]# mysqldump -uroot -p111111
imployee_salary IT_salary IT_desc > /opt/salary.sql
```

（2）使用 mysqldump 命令对单个库进行完全备份，命令格式如下：

```
mysqldump -u 用户名 -p [ 密码 ] [ 选项 ] [ 数据库名 ] > / 备份路径 / 备份文件名
```

下面演示了对库 imployee_salary 进行备份，备份文件是 /opt/salary2.sql。

```
[root@www mysql]# mysqldump -uroot -p111111 imployee_salary  > /opt/salary2.sql
```

（3）使用 mysqldump 命令对多个库进行完全备份，命令格式如下：

```
mysqldump -u 用户名 -p [ 密码 ] [ 选项 ]  --databases 库名 1[ 库名 2]... > / 备份路径 / 备份文件名
```

下面演示了对库 imployee_salary 和 test 进行备份，备份文件是 /opt/salary3.sql。

```
[root@www mysql]# mysqldump -uroot -p111111 --databases imployee_salary  test > /opt/salary3.sql
```

（4）使用 mysqldump 命令对所有库进行完全备份，命令格式如下：

```
mysqldump -u 用户名 -p [ 密码 ] [ 选项 ]  --all-databases > / 备份路径 / 备份文件名
```

下面演示了对所有库进行备份，备份文件是 /opt/salary4.sql。

```
[root@www mysql]# mysqldump -uroot -p111111 --all-databases> /opt/salary4.sql
```

（5）使用 mysqldump 命令也可以直接备份表结构，命令格式如下：

```
mysqldump -u 用户名 -p [ 密码 ] -d 数据库名 表名  > / 备份路径 / 备份文件名
```

只是多了一个 -d 选项，下面演示了对库 imployee_salary 中的表 IT_salary、IT_desc 进行表结构备份，备份文件是 /opt/salary5.sql。

```
[root@www mysql]# mysqldump -uroot -p111111 -d imployee_salary IT_salary IT_desc>
    /opt/salary5.sql
```

（6）下面分析 mysqldump 工具生成的备份文件的内容。

```
[root@www]# mysqldump -uroot -p111111 -d imployee_salary > /opt/salary6.sql
 [root@www]# cat /opt/salary6.sql
DROP TABLE IF EXISTS 'IT_desc';                    // 步骤 1
CREATE TABLE 'IT_desc' (                           // 步骤 2
 'descript' text,
 FULLTEXT KEY 'descript' ('descript')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
DROP TABLE IF EXISTS 'IT_salary';                  // 步骤 3
CREATE TABLE 'IT_salary' (                         // 步骤 4
 ' 岗位类别 ' char(48) DEFAULT NULL,
 ' 姓名 ' char(16) DEFAULT NULL,
 ' 年龄 ' int(11) DEFAULT NULL,
 ' 员工 ID' int(11) NOT NULL DEFAULT '0',
 ' 学历 ' char(16) DEFAULT NULL,
 ' 年限 ' int(11) DEFAULT NULL,
 ' 薪资 ' int(11) DEFAULT NULL,
 PRIMARY KEY (' 员工 ID')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

上面使用 -d 选项，说明只保存数据库的表结构，因为库中只有两个表，步骤 1 删除了 IT_desc 表，步骤 2 对它进行了创建操作，步骤 3 删除了 IT_salary 表，步骤 4 创

建了 IT_salary 表。删除和创建都是普通的 MySQL 操作命令，任何一条在 MySQL 中都是可以执行的语句，有了这些语句就可以创建出和现在的表结构相同的表。

如果不使用 -d 选项，会把数据也进行备份，下面看看数据是如何保存的。

```
[root@www mysql]# mysqldump -uroot -p111111  imployee_salary > /opt/salary7.sql
[root@www mysql]# cat /opt/salary7.sql
DROP TABLE IF EXISTS 'IT_desc';
CREATE TABLE 'IT_desc' (
 'descript' text,
 FULLTEXT KEY 'descript' ('descript')
) ENGINE=MyISAM DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS 'IT_salary';
CREATE TABLE 'IT_salary' (
 ' 岗位类别 ' char(48) DEFAULT NULL,
 ' 姓名 ' char(16) DEFAULT NULL,
 ' 年龄 ' int(11) DEFAULT NULL,
 ' 员工 ID' int(11) NOT NULL DEFAULT '0',
 ' 学历 ' char(16) DEFAULT NULL,
 ' 年限 ' int(11) DEFAULT NULL,
 ' 薪资 ' int(11) DEFAULT NULL,
 PRIMARY KEY (' 员工 ID')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
INSERT INTO 'IT_salary' VALUES (' 网络工程师 ',' 孙空武 ',27,11,' 本科 ',3,4800),
    ('Java 工程师 ',' 关园 ',38,14,' 大专 ',10,16000);
```

可以看到与使用 -d 参数的差别是最后出现了 insert into 语句，往数据库中插入了两条数据，也就是说，数据的备份是保存的插入语句操作。

通过上面的分析，很容易理解，备份的本质就是保存数据库当前表结构的创建语句和现有数据的插入语句，有了它们可以方便地恢复数据库的状态。

4.2 MySQL 完全恢复

前面介绍了数据库的完全备份，使用 mysqldump 工具可以把数据库的创建语句保存在备份的脚本文件中，当数据出现错误时，可以使用以下几种方式对它进行恢复操作。

4.2.1 恢复整库操作

当需要恢复整库的时候，可以使用 scource 命令和 mysql 命令。

1. source 命令整库恢复

登录 MySQL 后，可以使用 source 命令进行整库恢复，命令格式如下：

```
source 备份脚本路径
```

下面演示恢复过程。

（1）首先对库 imployee_salary 进行备份操作。

```
[root@www]# mysqldump -uroot -p111111 imployee_salary /opt/salary8.sql
```

（2）假设数据损坏，删除数据库 imployee_salary。

```
[root@www mysql]# mysql -uroot -p111111
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> drop database imployee_salary;
Query OK, 2 rows affected (0.05 sec)
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

（3）登录 MySQL，使用 source 命令恢复库 imployee_salary。

```
mysql> source /opt/salary8.sql
// 省略内容
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| imployee_salary    |                    //imployee_salary 恢复成功
| mysql              |
| test               |
+--------------------+
mysql> use imployee_salary;
Database changed
mysql> select * from IT_salary;
+----------------+-----------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名      | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+-----------+--------+----------+--------+--------+--------+
| 网络工程师     | 孙空武    |   27 |     11 | 本科   |    3 |  4800 |
| Java 工程师    | 关园      |   38 |     14 | 大专   |   10 | 16000 |
+----------------+-----------+--------+----------+--------+--------+--------+
2 rows in set (0.00 sec)
+--------------------+
```

使用 source 恢复整库后，库 imployee_salary 重新可以使用，表 IT_salary 里的数

据也恢复了。

2. MySQL 命令整库恢复

在不登录 MySQL 的情况下，可以使用 mysql 命令直接恢复整库，下面演示恢复过程。

（1）首先对库 imployee_salary 进行备份操作。

```
[root@www]# mysqldump -uroot -p111111 imployee_salary > /opt/salary8.sql
```

（2）假设数据损坏，删除数据库 imployee_salary。

```
[root@www mysql]# mysql -uroot -p111111
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> drop database imployee_salary;
Query OK, 2 rows affected (0.05 sec)
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

（3）不登录 MySQL，使用 mysql 命令恢复库 imployee_salary，命令格式如下：

```
mysql -u 用户名 -p [ 密码 ] < 库备份脚本的路径
```

此时库 imployee_salary 已经被删除了，需要先创建再进行恢复操作，否则会报错。

```
[root@www ~]# mysql -uroot -p111111
mysql> create database imployee_salary;
mysql> exit;
[root@www mysql]# mysql -uroot -p111111 imployee_salary < /opt/salary8.sql
[root@www mysql]# mysql -uroot -p111111
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| imployee_salary    |                    //imployee_salary 恢复成功
| mysql              |
| test               |
+--------------------+
mysql> use imployee_salary;
Database changed
mysql> select * from IT_salary;
```

```
+----------------+----------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名     | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+----------+--------+----------+--------+--------+--------+
| 网络工程师     | 孙空武   |   27 |     11 | 本科   |    3 |  4800 |
| Java 工程师    | 关园     |   38 |     14 | 大专   |   10 | 16000 |
+----------------+----------+--------+----------+--------+--------+------
```

使用 mysql 命令恢复整库后，库 imployee_salary 重新可以使用，表 IT_salary 里的数据也恢复了，说明恢复是成功的。

4.2.2　恢复表操作

恢复数据表同样可以使用 source 和 mysql 命令操作。

1. 使用 source 命令恢复表

使用 source 命令恢复表的操作与恢复库的操作方式相同，下面介绍这种方式。

（1）首先对表 IT_salary 进行备份操作。

```
[root@www mysql]# mysqldump -uroot -p111111 imployee_salary IT_salary > /opt/salary9.sql
```

（2）假设数据损坏，删除数据库 imployee_salary 中的表 IT_salary。

```
mysql> use imployee_salary;
Database changed
mysql> show tables;
+--------------------------+
| Tables_in_imployee_salary |
+--------------------------+
| IT_desc                  |
| IT_salary                |
+--------------------------+
2 rows in set (0.01 sec)
mysql> select * from IT_desc;
Empty set (0.01 sec)

mysql> select * from IT_salary;
+----------------+----------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名     | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+----------+--------+----------+--------+--------+--------+
| 网络工程师     | 孙空武   |   27 |     11 | 本科   |    3 |  4800 |
| Java 工程师    | 关园     |   38 |     14 | 大专   |   10 | 16000 |
+----------------+----------+--------+----------+--------+--------+------
mysql> drop table IT_salary;
Query OK, 0 rows affected (0.01 sec)
```

（3）登录 MySQL，使用 source 命令恢复表，命令格式如下：

```
[root@www mysql]# mysql -uroot -p111111
mysql> use imployee_salary;
Database changed
mysql> source /opt/salary9.sql;
mysql> show tables;
+--------------------------+
| Tables_in_imployee_salary |
+--------------------------+
| IT_desc                  |
| IT_salary                |
+--------------------------+
mysql> select * from IT_salary;
+----------------+----------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名     | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+----------+--------+----------+--------+--------+------
| 网络工程师     | 孙空武   |   27 |     11 | 本科   |    3 |  4800 |
| Java 工程师    | 关园     |   38 |     14 | 大专   |   10 | 16000 |
+----------------+----------+--------+----------+--------+--------+------
```

使用 source 命令恢复表后，表 IT_salary 里的数据也恢复了，说明恢复是成功的。

2. 使用 mysql 命令恢复表

使用 mysql 命令恢复表的操作时，当备份文件中只包含表的备份，而不包括创建库的语句时，必须指定库名，且目标库必须存在，下面介绍这种方法。

（1）首先对表 IT_salary 进行备份操作。

```
[root@www mysql]# mysqldump -uroot -p111111 imployee_salary IT_salary > /opt/salary9.sql
```

（2）假设数据损坏，删除数据库 imployee_salary 中的表 IT_salary。

```
mysql> use imployee_salary;
Database changed
mysql> show tables;
+--------------------------+
| Tables_in_imployee_salary |
+--------------------------+
| IT_desc                  |
| IT_salary                |
+--------------------------+
2 rows in set (0.01 sec)
mysql> select * from IT_desc;
Empty set (0.01 sec)

mysql> select * from IT_salary;
+----------------+----------+--------+----------+--------+--------+--------+
| 岗位类别       | 姓名     | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+----------------+----------+--------+----------+--------+--------+--------+
| 网络工程师     | 孙空武   |   27 |     11 | 本科   |    3 |  4800 |
| Java 工程师    | 关园     |   38 |     14 | 大专   |   10 | 16000 |
```

```
+-----------------+-----------+--------+----------+--------+--------+------
mysql> drop table IT_salary;
Query OK, 0 rows affected (0.01 sec)
```

（3）不登录 MySQL，使用 mysql 命令恢复表，命令格式如下：

```
mysql -u 用户名 -p [ 密码 ] 库名 < 库备份脚本的路径
[root@www]# mysql -uroot -p111111 imployee_salary < /opt/salary9.sql
[root@www mysql]# mysql -uroot -p111111
mysql> use imployee_salary;
mysql> show tables;
+-------------------------+
| Tables_in_imployee_salary |
+-------------------------+
| IT_desc                 |
| IT_salary               |
+-------------------------+
mysql> select * from IT_salary;
+-----------------+-----------+--------+----------+--------+--------+--------+
| 岗位类别        | 姓名      | 年龄   | 员工 ID  | 学历   | 年限   | 薪资   |
+-----------------+-----------+--------+----------+--------+--------+------
| 网络工程师      | 孙空武    |   27 |     11 | 本科   |    3 |  4800 |
| Java 工程师     | 关园      |   38 |     14 | 大专   |   10 | 16000 |
+-----------------+-----------+--------+----------+--------+--------+------
```

需要具体指定是哪个库，使用 mysql 命令恢复表后，表 IT_salary 中的数据也恢复了，说明恢复是成功的。

4.2.3　MySQL 备份思路

MySQL 需要定期实施备份，制定合适的备份计划或者策略，并严格遵守。除了进行完全备份，开启 MySQL 服务器的日志功能也很重要，完全备份加上日志，可以对 MySQL 进行最大化还原。

备份文件的名字需要使用统一和易理解的名称，不要使用 backup1、backup2 等这样没有意义的名字，推荐使用库名或者表名加上时间的命名规则，在需要恢复数据库的时候很容易定位到相应的备份文件。

4.2.4　完全备份恢复案例

生产环境中数据库备份需要制定计划，以保证数据的安全。本节通过以下案例讲解备份在生产环境中的应用 。

1. 需求描述

北京移电通信公司的用户信息数据库为 client，用户资费表为 user_info，表结构如表 4-2 所示。

表 4-2　表结构

身份证	姓名	性别	用户 id	资费
0000001	孙空武	男	011	100
0000002	蓝凌	女	012	98
0000003	姜纹	女	013	12
0000004	关园	男	014	38
0000005	罗中昆	男	015	39

需求如下：

（1）请为该公司制定合理的备份策略。

（2）依据所制定的策略备份数据。

（3）模拟数据丢失进行数据恢复。

2. 实现备份工作

（1）创建数据库 client，并在该库中建立 user_info 表，插入用户信息。

```
[root@www mysql]# mysql -uroot -p111111
mysql> create table user_info( 身份证 int(48), 姓名 varchar(20), 性别 varchar(20),
    用户 id 号 int(20) , 资费 int (48));
mysql> desc user_info;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| 身份证     | int(48)     | YES  |     | NULL    |       |
| 姓名       | varchar(20) | YES  |     | NULL    |       |
| 性别       | varchar(20) | YES  |     | NULL    |       |
| 用户 id    | int(20)     | YES  |     | NULL    |       |
| 资费       | int(48)     | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
mysql> insert into user_info values('0000001',' 孙空武 ',' 男 ',011,100);
mysql> insert into user_info values('0000002',' 蓝凌 ',' 女 ',012,98);
mysql> insert into user_info values('0000003',' 姜纹 ',' 女 ',013,12);
mysql> insert into user_info values('0000004',' 关园 ',' 男 ',014,38);
mysql> insert into user_info values('0000005',' 罗中昆 ',' 男 ',015,39);
mysql> select * from user_info;
+----------+----------+--------+-------------+--------+
| 身份证   | 姓名     | 性别   | 用户 id 号  | 资费   |
+----------+----------+--------+-------------+--------+
|       1  | 孙空武   | 男     |          11 |    100 |
|       2  | 蓝凌     | 女     |          12 |     98 |
|       3  | 姜纹     | 女     |          13 |     12 |
|       4  | 关园     | 男     |          14 |     38 |
|       5  | 罗中昆   | 男     |          15 |     39 |
```

```
+-----------+-----------+--------+-------------+--------+
5 rows in set (0.00 sec)
```

（2）制定合适的备份策略，如每天全备则使用 mysqldump 备份数据。

```
[root@www]# mysqldump -uroot -p111111 client user_info >/opt/client-user_info-$(date +%F).sql
```

（3）删除库 client，然后恢复数据，需要注意的是，需重建新的数据库 client，否则会报错。

```
[root@www]# mysql -uroot -p111111
mysql> drop database client;
mysql> exit;
[root@www]# mysql -uroot -p111111 client< /opt/client-user_info-2016-06-28.sql
ERROR 1046 (3D000) at line 22: No database selected
[root@www]# mysql -uroot -p111111
mysql> create database client;
mysql> exit;
[root@www]# mysql -uroot -p111111 client< /opt/client-user_info-2016-06-28.sql
```

（4）查看数据库内容，确定恢复成功。

```
[root@www]# mysql -uroot -p111111
mysql> use client;
Database changed
mysql> select * from user_info;
+-----------+-----------+--------+-------------+--------+
| 身份证    | 姓名      | 性别   | 用户 id     | 资费   |
+-----------+-----------+--------+-------------+--------+
|       1   | 孙空武    | 男     |      11     |   100  |
|       2   | 蓝凌      | 女     |      12     |    98  |
|       3   | 姜纹      | 女     |      13     |    12  |
|       4   | 关园      | 男     |      14     |    38  |
|       5   | 罗中昆    | 男     |      15     |    39  |
+-----------+-----------+--------+-------------+--------+
5 rows in set (0.00 sec)
```

本章总结

- 从物理与逻辑的角度，备份可分为物理备份（冷备份、热备份）、逻辑备份。
- 从数据库的备份策略角度，备份可分为完全备份、差异备份和增量备份。
- MySQL 中使用 mysqldump 工具备份，它生成的是 SQL 的脚本文件。
- 恢复数据使用 mysql、source 命令。
- 备份可以针对整库、某些库或某些表进行备份。

本章作业

1．举例说明完全备份、差异备份和增量备份的区别。

2．创建数据库 imployee_salary1 和 imployee_salary2，在这两个库中创建相同的表 IT_salary，并插入几条数据，对这两个库进行多库的完全备份，然后删除库 imployee_salary1，对它进行恢复操作。

3．用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



第5章

MySQL 增量备份与恢复

技能目标

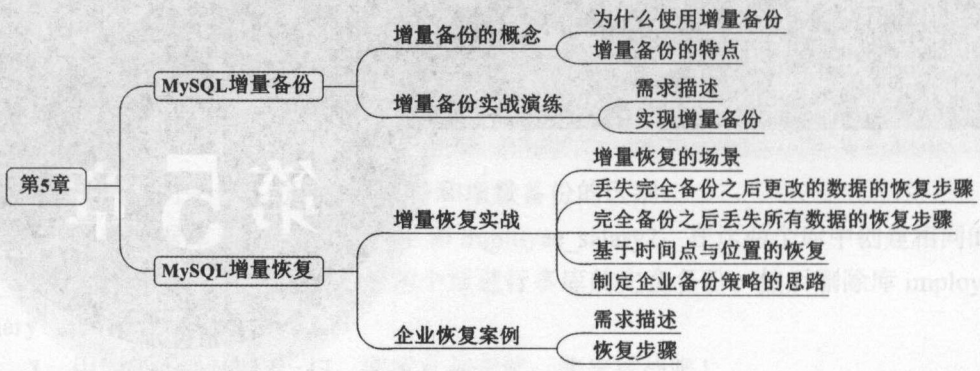
- 理解 MySQL 增量备份的概念
- 理解 MySQL 增量备份与恢复的方法

本章导读

本章介绍 MySQL 增量备份和恢复的方法，它可以在完全备份的基础上，减少备份文件的大小，加快备份和恢复的速度。

知识服务





5.1 MySQL 增量备份

5.1.1 增量备份的概念

1. 为什么使用增量备份

前面章节讲到了完全备份有两种方式，一种是使用 tar 打包数据文件，另一种是使用 mysqldump 进行完全备份。完全备份存在的问题很容易看到，每次都是把所有的数据内容进行备份，备份数据中有大量的重复数据，并且完全备份的时间与恢复的时间很长。解决完全备份存在的问题就是使用增量备份的方式，增量备份就是备份自上一次备份之后增加或改变的文件或者内容。

2. 增量备份的特点

增量备份的优点是没有重复数据，备份量不大，时间短。缺点也很明显，需要上次完全备份及完全备份之后所有的增量备份才能恢复，而且对所有增量备份进行逐个反推恢复，操作较为繁锁。

MySQL 没有提供直接的增量备份方法，但是可以通过 MySQL 的二进制日志（binary logs）间接实现增量备份。二进制日志对备份的意义如下：

（1）二进制日志保存了所有更新或者可能更新数据库的操作。

（2）二进制日志在启动 MySQL 服务器后开始记录，并在文件达到 max_binlog_size 所设置的大小或者接收到 flush logs 命令后重新创建新的日志文件。

（3）只需要定时执行 flush logs 方法重新创建新的日志，生成二进制文件序列，并及时把这些日志保存到安全的地方就完成了一个时间段的增量备份。

5.1.2 增量备份实战演练

下面是生产环境中的案例，对它进行增量备份操作。

1. 需求描述

北京移电通信公司的用户信息数据库为 client，用户资费表为 user_info。

需求如下：

（1）请为该公司每周进行完全备份。

（2）每天为该公司进行增量备份。

（3）新增加的用户信息如表 5-1 所示。

表 5-1　表结构

身份证	姓名	性别	用户 id	资费
0000006	张 1	男	016	100
0000007	张 2	女	017	98
0000008	张 3	女	018	12
0000009	张 4	男	019	38
00000010	张 5	男	020	39

2. 实现增量备份

（1）要进行 MySQL 增量备份，首先要开启二进制日志功能。开启二进制日志功能有两种方法：

1）在 MySQL 配置文件的 [mysqld] 项中加入 log-bin=filepath 项，如 log-bin=mysql-bin，然后重启 mysqld 服务。

```
[root@www ~]# vi /etc/my.cnf
log-bin=mysql-bin                    // 添加项
[root@www ~]# /etc/init.d/mysqld restart
Shutting down MySQL....
Starting MySQL...
```

2）使用 mysqld --log-bin=filepath 重新启动 MySQL 服务。

```
[root@www ~]# /etc/init.d/mysqld restart --log-bin=mysql-bin
Shutting down MySQL....
Starting MySQL...
```

（2）确定字符集使用的是 utf-8 的形式，避免出现乱码问题。

```
mysql> show variables like '%char%';
+--------------------------+--------------------------------+
| Variable_name            | Value                          |
+--------------------------+--------------------------------+
| character_set_client     | utf8                           |
| character_set_connection | utf8                           |
| character_set_database   | utf8                           |
| character_set_filesystem | binary                         |
```

```
| character_set_results    | utf8                          |
| character_set_server     | utf8                          |
| character_set_system     | utf8                          |
| character_sets_dir       | /usr/local/mysql/share/charsets/ |
+--------------------------+-------------------------------+
```

（3）创建备份文件保存的文件夹 /opt/bak_sql，备份文件都保存在这个文件夹中。

```
[root@www ~]# mkdir /opt/bak_sql
```

（4）每周选择服务器负载较轻的时间段，或者用户访问较少的时间段进行备份。使用 mysqldump 命令，可以对数据库或表进行备份。

1）使用 mysqldump 备份 client 库的 user_info 表。

```
[root@www]#mysqldump -uroot -p111111 client user_info> /opt/bak_sql/client_userinfo_$(date +%F).sql
[root@www]# ls /opt/bak_sql
client_userinfo_2016-06-29.sql
```

2）使用 mysqldump 备份 client 库。

```
[root@www~]# mysqldump -uroot -p111111 --databases client > /opt/bak_sql/client_$(date +%F).sql
[root@www~]# ls /opt/bak_sql/
client_2016-06-29.sql  client_userinfo_2016-06-29.sql
```

（5）每天进行增量备份操作，使用 mysqladmin 的选项 flush-logs 生成新的二进制文件，这样在插入新的数据后，新的二进制文件对应的就是数据库的变化的内容。下面操作可以看到新生成的二进制文件。

```
[root@www ~]# cd /usr/local/mysql/data/
[root@www data]# ls
mysql-bin.000042
[root@www data]# mysqladmin -uroot -p111111 flush-logs
[root@www data]# ls
mysql-bin.000042
mysql-bin.000043
```

生成之前 mysql-bin 文件最大的编号是 42，生成后最大编号是 43，之后的数据库操作会保存到编号 43 的二进制文件中。

（6）插入新的数据，以模拟数据的增加或者变更。

```
mysql> insert into user_info values('0000006',' 张 1',' 男 ',016,100);

mysql> insert into user_info values('0000007',' 张 2',' 女 ',017,98);

mysql> insert into user_info values('0000008',' 张 3',' 女 ',018,12);

mysql> insert into user_info values('0000009',' 张 4',' 男 ',019,38);

mysql> insert into user_info values('00000010',' 张 5',' 男 ',020,39);
```

（7）此时的数据库变化保存在编号 43 的二进制文件中，但是后面数据库再发生变化，编号 43 的文件还会产生变化，所以第二天可以再次执行 flush logs 生成新的二进制文件，确保编号 43 的文件不再改变，它只保存当天的数据库变化的内容。

```
[root@www data]# mysqladmin -uroot -p111111 flush-logs
[root@www data]# ls
mysql-bin.000042
mysql-bin.000043
mysql-bin.000044
```

使用 mysqlbinlog 命令可以查看二进制文件的内容，里面保存了插入数据的语句。

```
[root@www data]# mysqlbinlog mysql-bin.000043
// 省略内容
insert into user_info values('0000006',' 张 1',' 男 ',016,100)
insert into user_info values('0000007',' 张 2',' 女 ',017,98)
insert into user_info values('0000008',' 张 3',' 女 ',018,12)
insert into user_info values('0000009',' 张 4',' 男 ',019,38)
insert into user_info values('00000010',' 张 5',' 男 ',020,39)
```

（8）把二进制文件复制到保存备份的目录，完成增量备份。

```
[root@www data]# cp mysql-bin.000043 /opt/bak_sql/
[root@www data]# ls /opt/bak_sql
client_2016-06-29.sql  client_userinfo_2016-06-29.sql  mysql-bin.000043
```

下面对增量备份原理做个总结：首先是每周使用 mysqldump 完全备份整库，在整库备份的时间点之后，每天使用 flush logs 生成新的二进制文件，二进制文件保存的是每天对数据库的操作变化的内容，内容是不重复的。所以每周的备份整库加上每天的二进制备份文件，相当于是数据库现在的数据状态。

5.2　MySQL 增量恢复

增量恢复比完全恢复操作更为繁锁，每个增量备份都是单独的个体，数据不重复，需要控制的更加精确。

5.2.1　增量恢复实战

1. 增量恢复的场景

当数据发生错误时，应根据实际情况选择使用完全备份恢复，还是增量恢复。增量恢复的场景是：

（1）人为的 SQL 语句破坏了数据库。

（2）在进行下一次全备之前发生系统故障导致数据库数据丢失。

（3）在主从架构中，主库数据发生了故障。

根据数据丢失的情况可以分为两类：

（1）只丢失了完全备份之后更改的数据。

（2）完全备份之后丢失所有数据。

2. 丢失完全备份之后更改的数据的恢复步骤

当完全备份之后更改的数据丢失，需要把完全备份之后的所有增量备份文件逐个恢复，步骤如下：

（1）首先做一个完全备份操作，确保生成了完全备份的 sql 文件。

```
[root@www ~]# mysqldump -uroot -p111111 client > /opt/bak_sql/client_$(date +%F).sql
[root@www ~]# ls /opt/bak_sql/client_
client_2016-06-30.sql
```

（2）使用 flush-logs 生成新的二进制日志文件，用以保存之后的数据库操作语句。

```
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
[root@www ~]# ls /usr/local/mysql/data
// 省略内容
mysql-bin.000044
mysql-bin.000045
```

生成的编号为 45 的二进制文件保存之后的数据库操作语句。

（3）向数据库插入 1 条数据，执行一次 flush-logs 操作，生成新的二进制增量备份文件。

```
[root@www ~]# mysql -uroot -p111111
mysql> use client;
mysql> insert into user_info values('00000011',' 李 1',' 男 ',016,100);
mysql> exit;
Bye
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
[root@www ~]# ls /usr/local/mysql/data
mysql-bin.000044
mysql-bin.000045
mysql-bin.000046
```

编号为 45 的二进制日志文件中保存了插入语句。再次向数据库插入 1 条数据，语句会保存在编号为 46 的二进制文件中，然后再执行分割日志文件命令。

```
[root@www ~]# mysql -uroot -p111111
mysql> use client;
mysql> insert into user_info values('00000012',' 李 2',' 男 ',017,98);
Mysql> exit;
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
```

（4）使用 delete 删除插入的两条数据，也就是假设完全备份后的数据丢失了。

```
[root@www ~]# mysql -uroot -p111111
mysql> use client;
delete from user_info where 身份证 ='00000011';
delete from user_info where 身份证 ='00000012';
```

（5）使用二进制文件进行恢复操作，需要注意的是恢复的顺序，要先恢复最先生成的二进制文件，然后依次执行。

```
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000045| mysql -uroot -p111111
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000046| mysql -uroot -p111111
```

查看数据库内容，完全备份后新插入的 2 条数据找回来了，说明数据恢复成功。

```
mysql> use client;
Database changed
mysql> select * from user_info;
+-----------+-----------+--------+-------------+--------+
| 身份证    | 姓名      | 性别   | 用户 id     | 资费   |
+-----------+-----------+--------+-------------+--------+
|         1 | 孙空武    | 男     |          11 |    100 |
|         2 | 蓝凌      | 女     |          12 |     98 |
|         3 | 姜纹      | 女     |          13 |     12 |
|         4 | 关园      | 男     |          14 |     38 |
|         5 | 罗中昆    | 男     |          15 |     39 |
|         6 | 张 1      | 男     |          16 |    100 |
|         7 | 张 2      | 女     |          17 |     98 |
|         8 | 张 3      | 女     |          18 |     12 |
|         9 | 张 4      | 男     |          19 |     38 |
|        10 | 张 5      | 男     |          20 |     39 |
|        11 | 李 1      | 男     |          16 |    100 |
|        12 | 李 2      | 男     |          17 |     98 |
```

3. 完全备份之后丢失所有数据的恢复步骤

当完全备份和增量备份之后，所有的数据丢失，需要把完全备份和所有增量备份文件逐个恢复，这里演示对库 client 的恢复，对表的恢复操作方式相同。步骤如下：

（1）首先做一个完整的备份操作，确保生成了完全备份的 sql 文件。

```
[root@www ~]# mysqldump -uroot -p111111 client > /opt/bak_sql/client_$(date +%F).sql
[root@www ~]# ls /opt/bak_sql/client_
client_2016-06-30.sql
```

（2）使用 flush-logs 生成新的二进制日志文件，用以保存之后的数据库操作语句。

```
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
[root@www ~]# ls /usr/local/mysql/data
// 省略内容
mysql-bin.000044
mysql-bin.000045
```

生成的编号为 45 的二进制文件保存之后的数据库操作语句。

（3）向数据库插入 1 条数据，执行一次 flush-logs 操作，生成新的二进制增量备份文件。

```
[root@www ~]# mysql -uroot -p111111
mysql> use client;
mysql> insert into user_info values('00000011',' 李 1',' 男 ',016,100);
mysql> exit;
Bye
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
[root@www ~]# ls /usr/local/mysql/data
mysql-bin.000044
mysql-bin.000045
mysql-bin.000046
```

编号为 45 的二进制日志文件中就会保存插入语句。再次向数据库插入 1 条数据，语句会保存在编号为 46 的二进制文件中，然后执行分割二进制文件命令。

```
[root@www ~]# mysql -uroot -p111111
mysql> use client;
mysql> insert into user_info values('00000012',' 李 2',' 男 ',017,98);
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
```

（4）使用 drop 删除表 user_info，也就是假设完全备份前 user_info 的数据和完全备份后的数据都丢失了。

```
[root@www ~]# mysql -uroot -p111111
mysql> use client;
Database changed
mysql> drop user_info;
```

（5）先使用 mysql 命令进行完全备份的恢复操作。

```
[root@www ~]# mysql -uroot -p111111 client < /opt/bak_sql/client_2016-06-30.sql
[root@www ~]# mysql -uroot -p111111
mysql> select * from client.user_info;
+----------+----------+--------+------------+--------+
| 身份证   | 姓名     | 性别   | 用户 id    | 资费   |
+----------+----------+--------+------------+--------+
|        1 | 孙空武   | 男     |         11 |    100 |
|        2 | 蓝凌     | 女     |         12 |     98 |
|        3 | 姜纹     | 女     |         13 |     12 |
|        4 | 关园     | 男     |         14 |     38 |
|        5 | 罗中昆   | 男     |         15 |     39 |
|        6 | 张 1     | 男     |         16 |    100 |
|        7 | 张 2     | 女     |         17 |     98 |
|        8 | 张 3     | 女     |         18 |     12 |
|        9 | 张 4     | 男     |         19 |     38 |
```

```
|      10 | 张 5      | 男   |       20      |   39 |
```

被删除的表中的数据又可以查询出来，说明完全恢复是成功的。

（6）使用二进制文件进行增量恢复操作，需要注意的是恢复的顺序，要先恢复最先生成的二进制文件，然后依次执行。

```
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000045| mysql -uroot -p111111
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000046| mysql -uroot -p111111
```

查看数据库内容，增量备份的新插入的两条数据找回来了，说明数据恢复成功。

```
mysql> use client;
Database changed
mysql> select * from user_info;
+-----------+-----------+--------+-------------+--------+
| 身份证    | 姓名      | 性别   | 用户 id     | 资费   |
+-----------+-----------+--------+-------------+--------+
|       1   | 孙空武    | 男     |        11   |   100  |
|       2   | 蓝凌      | 女     |        12   |    98  |
|       3   | 姜纹      | 女     |        13   |    12  |
|       4   | 关园      | 男     |        14   |    38  |
|       5   | 罗中昆    | 男     |        15   |    39  |
|       6   | 张 1      | 男     |        16   |   100  |
|       7   | 张 2      | 女     |        17   |    98  |
|       8   | 张 3      | 女     |        18   |    12  |
|       9   | 张 4      | 男     |        19   |    38  |
|      10   | 张 5      | 男     |        20   |    39  |
|      11   | 李 1      | 男     |        16   |   100  |
|      12   | 李 2      | 男     |        17   |    98  |
```

4. 基于时间点与位置的恢复

利用二进制日志可实现基于时间点与位置的恢复，例如由于误操作删除了一张表，这时完全恢复是没有用的，因为日志里面还存在误操作的语句，我们需要的是恢复到误操作前的状态，然后跳过误操作的语句，再恢复后面操作的语句。

假定需要往数据库中插入两条数据，但由于误操作，两条插入语句中间删除了一条数据，而这条数据是不应该删除的。

```
[root@www ~]# mysqladmin -uroot -p111111 flush-logs     // 进行二进制文件分割
[root@www ~]# ls /usr/local/mysql/data
mysql-bin.000048                                        // 新的二进制文件编号为 48
[root@www ~]# mysql -uroot -p111111
mysql> use client;
mysql> insert into user_info values('00000013',' 赵 1',' 男 ',016,100);
mysql> delete from user_info where 身份证 ='00000012';
mysql> insert into client.user_info values('00000014',' 赵 2',' 男 ',017,98);
mysql> select * from user_info;
```

```
// 省略内容
|     11 | 李 1    | 男   |      16 |  100 |
|     13 | 赵 1    | 男   |      16 |  100 |
|     14 | 赵 2    | 男   |      17 |   98 |
```

编号为 48 的二进制文件中保存了正确的插入语句，同时也保存了不应该执行的删除语句。

```
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000048
// 省略内容
#160701  0:03:55        // 时间
insert into user_info values('00000013',' 赵 1',' 男 ',016,100)
# at 513                // 位置
#160701  0:04:09        // 时间
delete from user_info where 身份证 ='00000012'
# at 540                // 位置
#160701  0:04:18        // 时间
insert into client.user_info values('00000014',' 赵 2',' 男 ',017,98)
```

二进制文件中对语句的执行时间和位置也进行了保存，下面演示基于时间和位置的恢复方法。

（1）基于时间点的恢复，就是将某个起始时间的二进制日志导入数据库中，从而跳过某个发生错误的时间点实现数据的恢复。使用 mysqlbinlog 加上 --stop-datetime 选项，表示在哪个时间点结束，后面误操作的语句不执行，--start-datetime 选项表示执行后面的语句，结合使用它们就可以跳过误操作的语句，完成恢复工作。需要注意的是，二进制文件中保存的日期格式需要调整为用“-”分割。

```
[root@www]#mysqlbinlog --stop-datetime='16-07-01 0:04:09' /usr/local/mysql/data/mysql-
    bin.000048| mysql -uroot -p111111
[root@www]#mysqlbinlog --start-datetime='16-07-01 0:04:18' /usr/local/mysql/data/mysql-
    bin.000048| mysql -uroot -p111111
```

删除语句执行的时间点是 0:04:09，--stop-datetime 设置为 0:04:09 与它相同，表示删除语句之前的语句执行，也就是第一条插入语句执行，而删除语句及第二条插入语句不执行。--start-datetime 设置为 0:04:018 指定的是第二条插入语句执行的时间点，也就是执行第二条插入语句。所以只执行了两条插入语句，而误操作的删除语句不执行，达到了恢复数据的目的。

（2）基于位置的恢复，就是使用基于时间点的恢复，可能会出现在一个时间点里既同时存在正确的操作又存在错误的操作，基于位置是一种更为精确的恢复方式。

```
[root@www]#mysqlbinlog          --stop-position='513'
/usr/local/mysql/data/mysql-bin.000048| mysql -uroot -p111111
[root@www]#mysqlbinlog          --start-position='540'
/usr/local/mysql/data/mysql-bin.000048| mysql -uroot -p111111
```

第一条插入语句后面的位置是 513，--stop-position 设置为 513，表示执行第一条

插入语句后结束。第二条插入语句的位置是 540，--start-positon 设置为 540，表示执行第二条插入语句，跳过了误操作的删除语句，达到了恢复数据的目的。

5. 制定企业备份策略的思路

制定企业备份策略要根据企业数据库的实际读写的频繁性与数据的重要性进行。

（1）数据更新频繁，则应该进行较为频繁的备份。

（2）数据较为重要，则在有适当更新时进行备份。

（3）在数据库压力小的时段进行全备，如一周一次，然后每天增备。

根据公司的规模，中小公司可一天一次全备，大公司可每周一次全备，每天进行一次增备，并且尽量为企业实现主从复制架构。

5.2.2　企业恢复案例

1. 需求描述

北京移电通信公司的用户信息数据库为 client，用户资费数据表为 user_info，由于人为原因用户资费表 user_info 的数据已经完全损坏，如表 5-2 所示。请利用所做的备份为该公司进行数据恢复。

表 5-2　表结构

身份证	姓名	性别	用户 id	资费
0000006	null	男	XXX	null
0000007	XXX	女	017	98
0000008	XXX	女	XXX	null
0000009	XXX	男	019	38
00000010	XXX	男	XXX	null

2. 恢复步骤

（1）先对二进制日志进行分割处理，然后使用 mysql 命令进行完全备份的恢复操作。

```
[root@www ~]# mysqladmin -uroot -p111111 flush-logs
[root@www ~]# mysql -uroot -p111111 client < /opt/bak_sql/client_2016-06-30.sql
[root@www ~]# mysql -uroot -p111111
mysql> select * from client.user_info;
+----------+----------+--------+-------------+--------+
| 身份证   | 姓名     | 性别   | 用户 id     | 资费   |
+----------+----------+--------+-------------+--------+
|      1   | 孙空武   | 男     |          11 |    100 |
|      2   | 蓝凌     | 女     |          12 |     98 |
|      3   | 姜纹     | 女     |          13 |     12 |
|      4   | 关园     | 男     |          14 |     38 |
```

```
|    5    |罗中昆 |男    |      15      |   39|
|    6    |张 1   |男    |      16      |  100|
|    7    |张 2   |女    |      17      |   98|
|    8    |张 3   |女    |      18      |   12|
|    9    |张 4   |男    |      19      |   38|
|   10    |张 5   |男    |      20      |   39|
```

数据库中的数据是完全备份时的状态，说明完全恢复是成功的。

（2）使用二进制文件进行增量恢复操作，需要注意的是恢复的顺序，要先恢复最先生成的二进制文件，然后依次执行。

```
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000045| mysql -uroot -p111111
[root@www ~]# mysqlbinlog  /usr/local/mysql/data/mysql-bin.000046| mysql -uroot -p111111
```

查看数据库内容，增量备份的新插入的两条数据找回来了，说明数据恢复成功。

```
mysql> use client;
Database changed
mysql> select * from user_info;
+----------+----------+--------+-------------+--------+
|身份证    |姓名      |性别    |用户 id      |资费    |
+----------+----------+--------+-------------+--------+
|    1     |孙空武   |男      |      11     |  100   |
|    2     |蓝凌     |女      |      12     |   98   |
|    3     |姜纹     |女      |      13     |   12   |
|    4     |关园     |男      |      14     |   38   |
|    5     |罗中昆   |男      |      15     |   39   |
|    6     |张 1     |男      |      16     |  100   |
|    7     |张 2     |女      |      17     |   98   |
|    8     |张 3     |女      |      18     |   12   |
|    9     |张 4     |男      |      19     |   38   |
|   10     |张 5     |男      |      20     |   39   |
|   11     |李 1     |男      |      16     |  100   |
|   12     |李 2     |男      |      17     |   98   |
```

本章总结

MySQL 没有直接提供增量备份的工具，需要借助二进制日志文件进行操作。

- 使用分割日志的方式进行增量备份。
- 增量恢复需要根据日志文件的时间先后，逐个执行。
- 使用基于时间和位置的方式进行恢复，可以更精确地恢复数据。
- 大企业应每周做一次全备，每天做一次增量备份；中小企业应每天进行一次全备。

本章作业

1．简述 MySQL 增量备份与增量恢复的方法。

2．创建用户信息数据库 client，用户资费表 user_info，数据如表 5-3 所示。

表 5-3　表结构

身份证	姓名	性别	用户 id	资费
0000006	张 1	男	016	100
0000007	张 2	女	017	98
0000008	张 3	女	018	12
0000009	张 4	男	019	38
00000010	张 5	男	020	39

对库进行完全备份和增量备份，然后插入姓名为“李 1”“李 2”和“李 3”这三条数据。分别采用基于时间和基于位置的方式进行增量恢复操作，使表中只有“李 1”和“李 3”这两条数据。

3．用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



随手笔记



第6章

MySQL 存储引擎 MyISAM 和 InnoDB

技能目标

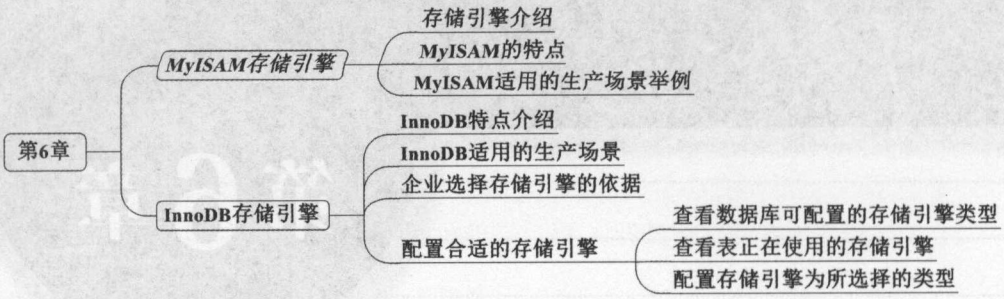
- 了解 MyISAM 存储引擎的特性
- 了解 InnoDB 存储引擎的特性

本章导读

存储引擎的实质就是如何实现存储数据，为存储的数据建立索引以及查询、更改、删除数据等技术实现的方法。不同于很多数据库系统仅仅选择一种类型的数据存储方式，MySQL 数据库支持多种不同的数据引擎，可以根据具体的要求，针对每一个表使用不同的数据引擎，摒弃了很多其他数据库采取的“一个尺码满足一切需求”的解决方案。正是这种灵活性，使得 MySQL 数据库如此受欢迎。

知识服务





6.1 MyISAM 存储引擎

6.1.1 存储引擎介绍

MySQL 中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力。通过选择不同的技术，能够获得额外的速度或者功能，从而改善应用的整体功能。这些不同的技术以及配套的相关功能在 MySQL 中被称作存储引擎。

MySQL 提供了多个不同的存储引擎，可以预先设置或者在 MySQL 服务器中启用。根据实际需要还可以分别选择使用于服务器、数据库或者表格的存储引擎，以便在后续检索这些信息等操作过程中提供最大的灵活性。例如，研究大量的临时数据时，可以选择内存存储引擎，能够在内存中存储所有的表格数据，也可以选择支持事务处理的数据引擎，以确保事务处理不成功时数据能够回退等。可以看出 MySQL 的核心就在其存储引擎上。

在熟悉存储引擎之前，先来了解 MySQL 的体系结构，这是在使用引擎时很重要的一环。MySQL 的体系架构如图 6.1 所示。

从这个体系结构中，可以清楚地看出 MySQL 是由 SQL 接口、解析器、优化器、缓存、存储引擎组成。

- Connectors：连接组件。
- Enterprise Management Service & Utilities：管理服务和控制组件。
- Connection Pool：连接池组件。
- SQL Interface：SQL 接口组件。
- Parser：查询分析器组件。
- Optimizer：优化器组件。
- Caches & Buffers：缓存组件。
- Pluggable Storage Engines：插件式存储引擎。
- File System：文件系统。

- File & Logs：存储文件和日志。

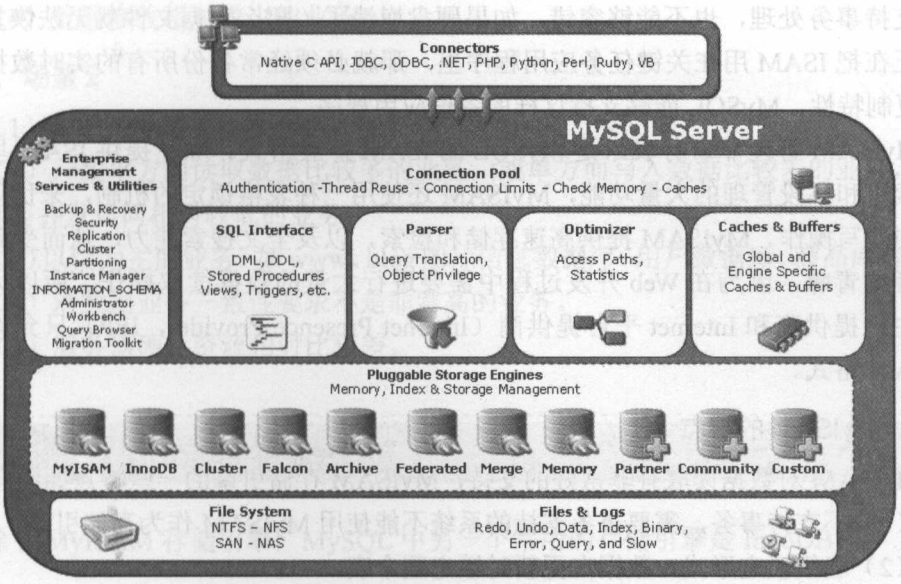


图 6.1　MySQL 的体系架构

从图 6.1 中可以看出来，MySQL 提供了一系列标准的管理和服务支持，这些标准和存储引擎无关，是每个数据库本身所必需的，如 SQL Interface、Parser、Optimizer 等。而存储引擎是底层物理结构的实现，每个开发者都可以按照自己的意愿来开发或者修改存储引擎的内容。

MySQL 支持插件式的表存储引擎，这种独有的插件式体系架构，让存储引擎有了依赖于应用的多样性。开发人员能够根据应用的不同特点，选择合适的存储引擎；如果存储引擎不能满足要求的话，还可以定制适合自己应用的存储引擎。eBay、Google、Facebook 等公司都对 MySQL 存储引擎做过改进，使其满足自己的业务特性，最为知名的是 MyISAM 和 InnoDB 存储引擎。

MySQL 系统中，存储引擎处于文件系统之上，在数据保存到数据文件之前会先传输到存储引擎，然后按照各个存储引擎的存储格式进行数据存储。使用这种存储引擎的主要优点在于，仅仅需要提供特殊应用所需的特性即可；数据库中的系统开销较小；具有更有效和更高效的数据库性能。

6.1.2　MyISAM 的特点

1. MyISAM 的介绍

MyISAM 存储引擎是 MySQL 关系数据库系统 5.5 版本之前默认的存储引擎，它的前身是 ISAM。ISAM 是一个定义明确且历经时间考验的数据表格管理方法，它在设

计之时就考虑到数据库被查询的次数要远大于更新的次数。因此，ISAM 执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM 的两个主要不足之处在于，它不支持事务处理，也不能够容错。如果硬盘崩溃了，那么数据文件就无法恢复了。如果正在把 ISAM 用在关键任务应用程序里，那就必须经常备份所有的实时数据，通过其复制特性，MySQL 能够支持这样的备份应用程序。

MyISAM 管理非事务表，是 MySQL 的 ISAM 扩展格式。除了提供 ISAM 里所没有的索引和字段管理的大量功能，MyISAM 还使用一种表格锁定的机制，来优化多个并发的读写操作。MyISAM 提供高速存储和检索，以及全文搜索能力，从而受到 Web 开发者的青睐，因为在 Web 开发过程中需要进行大量的数据读取操作，所以大多数虚拟主机提供商和 Internet 平台提供商（Internet Presence Provider，IPP）只允许使用 MyISAM 格式。

2. MyISAM 的特点

MyISAM 对数据读取有非常好的支持，MyISAM 存储引擎的一些特点如下：

（1）不支持事务，需要事务支持的系统不能使用 MyISAM 作为存储引擎。

（2）表级锁定形式，数据在更新时锁定整个表。

（3）数据库在读写过程中相互阻塞：

- 在数据写入的过程中阻塞用户对数据的读取。
- 在数据读取的过程中阻塞用户写入数据。

（4）可以通过 key_buffer_size 来设置缓存索引，提高访问的性能，减少磁盘 IO 的压力。

（5）采用 MyISAM 存储引擎进行数据单独写入或读取，速度较快且占用资源相对要少。

（6）MyISAM 存储引擎不支持外键约束，只支持全文索引。

（7）每个 MyISAM 在磁盘上存储成三个文件。每一个文件的名字均以表的名字开始，扩展名指出文件类型：

- .frm 文件存储表定义。
- 数据文件的扩展名为：.MYD（MYData）。
- 索引文件的扩展名为：.MYI（MYIndex）。

6.1.3 MyISAM 适用的生产场景举例

在生产场景中，需要根据场景的实际情况，选择合适的存储引擎，下面举两个例子，说明在什么场景中选择使用 MyISAM 存储引擎。

1. 场景 1

（1）公司的业务不需要事务的支持，这点是非常明确的，因为 MyISAM 没有事务。

（2）一般单方面读取数据比较多的业务，或单方面写入数据比较多的业务。因为

MyISAM 具有读写互相阻塞的特点，数据读写都比较频繁的场景不适合使用。

（3）使用读写并发访问相对较低的业务。

（4）数据修改相对较少的业务。

2. 场景 2

（1）公司的业务不需要事务的支持。

（2）一般单方面读取数据比较多的业务，或单方面写入数据比较多的业务，以及使用读写并发访问相对较低的业务。

（3）以读为主的业务，如 www、blog 图片信息数据库、用户数据库、商品库等业务。

（4）对数据业务一致性要求不是非常高的业务。

（5）服务器硬件资源相对比较差。

6.2　InnoDB 存储引擎

除了 MyISAM 存储引擎，MySQL 中另一个常用的存储引擎是 InnoDB。

6.2.1　InnoDB 特点介绍

InnoDB 存储引擎不同于 MyISAM 存储引擎，能够支持事务安全表（ACID 兼容），具有提交、回滚和崩溃恢复的能力。InnoDB 支持行级锁定，并且可以在 SELECT 语句中提供一个与 Oracle 风格类似的非锁定读，也支持 FOREIGN KEY 强制。种种特性使得 InnoDB 存储引擎的多用户部署和性能得以提升。InnoDB 存储引擎还支持 SQL 查询中将 InnoDB 类型的表与其他 MySQL 的表类型混合使用，甚至在同一个查询中也可以混合使用。

InnoDB 是为处理巨大数据量时的最大性能而设计的。它的 CPU 效率可能是任何其他基于磁盘的关系数据库引擎所不能匹敌的。 InnoDB 存储引擎被完全与 MySQL 服务器整合，为在内存中缓存数据和索引而维持它自己的缓冲池。 InnoDB 在一个表空间中存储它的表与索引，表空间可以包含数个文件（或原始磁盘分区）。这与 MyISAM 表不同，比如在 MyISAM 表中每个表被存储在分离的文件中。InnoDB 表可以是任何尺寸，即使在文件尺寸被限制为 2GB 的操作系统上也无影响。

对 InnoDB 的特点总结如下：

（1）支持事务，支持四个事务隔离级别。

（2）行级锁定，但是全表扫描仍然会是表级锁定。

（3）读写阻塞与事务隔离级别相关。

（4）具有非常高效的缓存特性，能缓存索引，也能缓存数据。

（5）表与主键以簇的方式存储。

（6）支持分区、表空间，类似 Oracle 数据库。

（7）支持外键约束，MySQL 5.5 以前不支持全文索引，5.5 版本以后支持。

（8）适合对硬件资源要求比较高的场合。

6.2.2 InnoDB 适用的生产场景

根据 InnoDB 的特点，在下面的场景中需要选择使用 InnoDB 存储引擎。

（1）业务需要事务的支持。

（2）行级锁定对高并发有很好的适应能力，但需要确保查询是通过索引来完成。

（3）业务数据更新较为频繁的场景，如论坛、微博等。

（4）业务数据一致性要求较高，例如银行业务。

（5）硬件设备内存较大，利用 InnoDB 较好的缓存能力来提高内存利用率，减少磁盘 IO 的压力。

6.2.3 企业选择存储引擎的依据

选择哪种存储引擎，应根据企业系统的实际需要，考虑每个存储引擎提供了哪些不同的核心功能及应用的场景。主要是依据以下几个方面来决定：

（1）存储引擎支持的字段和数据类型

所有引擎都支持通用的数据类型，但不是所有的引擎都支持其他的字段类型，如二进制对象。

（2）锁定类型

不同的存储引擎支持不同级别的锁定：

- 表锁定：MyISAM 支持。
- 行锁定：InnoDB 支持。

（3）索引的支持

建立索引在搜索和恢复数据库中的数据时能够显著提高性能。

不同的存储引擎提供不同的制作索引的技术。

有些存储引擎根本不支持索引。

（4）事务处理的支持

事务处理功能提供向表中更新和插入信息期间的可靠性。

可根据企业业务是否要支持事务选择存储引擎。

6.2.4 配置合适的存储引擎

选择好了合适的存储引擎之后，就可以修改为相应的存储引擎类型，修改的步骤如下：

（1）查看数据库可配置的存储引擎类型。

（2）查看表正在使用的存储引擎类型。

（3）配置存储引擎为所选择的类型。

下面详细介绍操作的步骤。

1. 查看数据库可配置的存储引擎类型

使用 show engines 可以查看 MySQL 默认使用的存储引擎。

```
mysql> show engines;
|Engine|Support|Transactions
|MRG_MYISAM| YES    |No
| InnoDB  | DEFAULT | YES
| MyISAM | YES      | No
| CSV    | YES      | No
| MEMORY | YES      | No
// 省略内容
```

其中，Engine 列表示当前版本的 MySQL 所支持的引擎类型；Support 列表示对应引擎是否能使用；Transactions 列表示对应引擎是否支持事务；Default 表示该引擎为当前默认存储引擎。

可以看到 InnoDB 是默认的存储引擎，它支持事务的操作。MyISAM 在当前版本的 MySQL 中是支持的，还有一些其他的存储引擎也是支持的，修改存储引擎时可以指定。

2. 查看表正在使用的存储引擎

查看表正在使用的存储引擎类型，有两种方式。

（1）使用 show table status 命令可以查看表正在使用的存储引擎，命令格式如下：

```
show table status  from 库名 where  name = ' 表名 ' ;
| Name     | Engine | Version | Row_format | Rows |
| user_info | InnoDB |     10 | Compact    |  13 |
// 省略内容
```

可以看到 user_info 表使用的存储引擎是 InnoDB。

（2）使用 show create 命令也可以查看表正在使用的存储引擎，命令格式如下：

```
show   create   table   表名;
mysql> use client;
Database changed
mysql> show create table user_info;
| Table     | Create Table
| user_info | CREATE TABLE 'user_info' (
 ' 身份证 ' int(48) DEFAULT NULL,
 ' 姓名 ' varchar(20) DEFAULT NULL,
 ' 性别 ' varchar(20) DEFAULT NULL,
 ' 用户 id' int(20) DEFAULT NULL,
```

```
  ' 资费 ' int(48) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

3. 配置存储引擎为所选择的类型

当需要选择非默认存储引擎时，修改存储引擎的方法有四种。

（1）使用 alter table 命令修改，命令格式如下：

```
alter table   table_name  engine= 引擎;
mysql> use client;
Database changed
mysql> show create table user_info;
| Table    | Create Table
| user_info | CREATE TABLE 'user_info' (
  ' 身份证 ' int(48) DEFAULT NULL,
  ' 姓名 ' varchar(20) DEFAULT NULL,
  ' 性别 ' varchar(20) DEFAULT NULL,
  ' 用户 id' int(20) DEFAULT NULL,
  ' 资费 ' int(48) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8      // 默认存储引擎是 InnoDB
mysql> alter table user_info engine=MyISAM;      // 修改存储引擎为 MyISAM
Query OK, 13 rows affected (0.05 sec)
Records: 13  Duplicates: 0  Warnings: 0
mysql> show create table user_info;
| Table    | Create Table
| user_info | CREATE TABLE 'user_info' (
  ' 身份证 ' int(48) DEFAULT NULL,
  ' 姓名 ' varchar(20) DEFAULT NULL,
  ' 性别 ' varchar(20) DEFAULT NULL,
  ' 用户 id' int(20) DEFAULT NULL,
  ' 资费 ' int(48) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8    // 修改成功
```

如果设置 engine=InnoDB，表的存储引擎就是 InnoDB。

```
mysql> alter table user_info engine=InnoDB;      // 修改存储引擎为 MyISAM
Query OK, 13 rows affected (0.05 sec)
Records: 13  Duplicates: 0  Warnings: 0
mysql> show create table user_info;
| Table    | Create Table
| user_info | CREATE TABLE 'user_info' (
  ' 身份证 ' int(48) DEFAULT NULL,
  ' 姓名 ' varchar(20) DEFAULT NULL,
  ' 性别 ' varchar(20) DEFAULT NULL,
  ' 用户 id' int(20) DEFAULT NULL,
  ' 资费 ' int(48) DEFAULT NULL
) ENGINE= InnoDB DEFAULT CHARSET=utf8          // 修改成功
```

（2）修改 MySQL 的配置文件 my.cnf，可以指定 default-storage-engine 选项设置

默认的存储引擎。

```
[root@www ~]# vi /etc/my.cnf
// 省略内容
[mysqld]
default-storage-engine=MyISAM
// 省略内容
```

这里指定默认存储引擎是 MyISAM，需要重新启动 MySQL 服务器，创建新表将使用 MyISAM 存储引擎。

```
[root@www ~]# /etc/init.d/mysqld restart
Shutting down MySQL.                                    [ 确定 ]
Starting MySQL.                                         [ 确定 ]
mysql> create database linuxfan;
Query OK, 1 row affected (0.01 sec)
mysql> use linuxfan;
Database changed
mysql> create table test (id int);
Query OK, 0 rows affected (0.03 sec)
mysql> show create table test;
| Table | Create Table
| test  | CREATE TABLE 'test' (
  'id' int(11) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |      // 新表的存储引擎是 MyISAM
```

默认指定的存储引擎是针对新建的表，对以前已经存在的表没有影响。前面表 user_info 的存储引擎是 InnoDB，并不会因为设置了默认存储引擎而改变。

```
mysql> show create table user_info;
| Table     | Create Table
| user_info | CREATE TABLE 'user_info' (
  ' 身份证 ' int(48) DEFAULT NULL,
  ' 姓名 ' varchar(20) DEFAULT NULL,
  ' 性别 ' varchar(20) DEFAULT NULL,
  ' 用户 id' int(20) DEFAULT NULL,
  ' 资费 ' int(48) DEFAULT NULL
) ENGINE= InnoDB DEFAULT CHARSET=utf8          //InnoDB 没有改变
```

（3）使用 create table 创建表时用 engine 指定存储引擎，会使用指定的存储引擎。现在数据库的默认引擎是 MyISAM，但创建新表时指定使用 InnoDB，则新表的存储引擎就是 InnoDB。

```
mysql> use client;
Database changed
mysql> create table test (id int) engine=InnoDB;
mysql> show create table test;
| Table | Create Table
```

```
| test  | CREATE TABLE 'test' (
  'id' int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8          // 新表存储引擎是 InnoDB
```

（4）使用 mysql_convert_table_format 命令可以批量转换存储引擎，命令格式如下：

```
mysql_convert_table_format --user=root  --password= 密码 --socket=/temp/mysql.sock  --engine
    = 引擎  库名  表名
```

命令参数如表 6-1 所示。

表 6-1　命令参数

参数	描述
--force	如果碰到错误强制进行转换
--host	转换的主机名
--password	当前进行转换用户的密码
--port	如果不是以 localhost 连接的话，请指定端口
--socket	SOCKET 文件存在的位置
--user	连接的用户名

使用 mysql_convert_table_format 转换存储引擎时，可能会遇到下面的问题，这里提供了解决方法。

```
[root@www ~]# mysql_convert_table_format --host=localhost --user=root --password=111111
      --socket=/tmp/mysql.sock --engine=MyISAM client test
Can't locate DBI.pm in @INC (@INC contains:
/usr/local/lib64/perl5
/usr/local/share/perl5
/usr/lib64/perl5/vendor_perl
/usr/share/perl5/vendor_perl
/usr/lib64/perl5
/usr/share/perl5 .)
at /usr/local/mysql/bin/mysql_convert_table_format line 20.
BEGIN failed--compilation aborted at /usr/local/mysql/bin/mysql_convert_table_format line 20.)
ENGINE=InnoDB DEFAULT CHARSET=utf8
```

当出现以上错误时，表示没有安装 perl，需要使用 yum 进行安装。

```
[root@www ~]# yum -y install perl-DBI
```

再次执行转换操作，又会报另一个错误：

```
[root@www ~]# mysql_convert_table_format --host=localhost --user=root --password=111111
      --socket=/tmp/mysql.sock --engine=MyISAM client test
install_driver(mysql) failed: Can't locate DBD/mysql.pm in @INC (@INC contains: /usr/local/lib64/
  perl5 /usr/local/share/perl5 /usr/lib64/perl5/vendor_perl /usr/share/perl5/vendor_perl /usr/lib64/perl5/
  usr/share/perl5 .) at (eval 3) line 3.
```

```
Perhaps the DBD::mysql perl module hasn't been fully installed,
or perhaps the capitalisation of 'mysql' isn't right.
Available drivers: DBM, ExampleP, File, Gofer, Proxy, Sponge.
 at /usr/local/mysql/bin/mysql_convert_table_format line 64
```

表示 perl 对 mysql 的操作模块没有安装，需要使用 yum 对它进行安装。

```
[root@www ~]# yum -y install perl-DBD-MySQL
```

现在可以正常使用 mysql_convert_table_format 命令进行存储引擎的转换操作。client 库中的 test 表的存储引擎现在是 InnoDB。

```
[root@www ~]# mysql -uroot -p111111
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> use client;
Database changed
mysql> show create table test;
| Table | Create Table
| test  | CREATE TABLE 'test' (
 'id' int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

使用 mysql_convert_table_format 命令把它转换为 MyISAM 存储引擎。

```
[root@www ~]# mysql_convert_table_format --host=localhost --user=root --password=111111
    --socket=/tmp/mysql.sock --engine=MyISAM client test
```

可以看到 test 的存储引擎已经是 MyISAM，说明转换成功。

```
[root@www ~]# mysql -uroot -p111111
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql> use client;
Database changed
mysql> show create table test;
| Table | Create Table
| test  | CREATE TABLE 'test' (
 'id' int(11) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
```

前面已经说过 mysql_convert_table_format 是批量转换，那么它能不能转换为 InnoDB 呢？现在指定 --engine=InnoDB，看看会有什么结果。

```
[root@www ~]# mysql_convert_table_format --host=localhost --user=root --password=111111
    --socket=/tmp/mysql.sock --engine=InnoDB client test
test already uses the 'MYISAM' engine;  Ignored
```

出现了一个提示消息“表 test 已经在使用 MYISAM 存储引擎了，忽略掉”。其实这是 mysql_convert_table_format 命令的一个 bug，也就是只能把表转换为 MyISAM，如果批量转换为其他存储引擎，需要对 mysql_convert_table_format 命令脚本进行修改。

mysql_convert_table_format 命令脚本在 MySQL 安装目录的 bin 下面，它的第 32

行要进行如下的修改：

```
[root@www bin]# vi /usr/local/mysql/bin/mysql_convert_table_format
$opt_help=$opt_version=$opt_verbose=$opt_force=0;
$opt_user=$opt_database=$opt_password=undef;
$opt_host="localhost";
$opt_socket="";
$opt_engine="MYISAM";
$opt_port=0;
$exit_status=0;
GetOptions(
 "e|engine|type=s"     => \$opt_engine,          //32 行
 "f|force"           => \$opt_force,
 "help|?"            => \$opt_help,
 "h|host=s"          => \$opt_host,
 "p|password=s"      => \$opt_password,
 "u|user=s"          => \$opt_user,
 "v|verbose"         => \$opt_verbose,
 "V|version"         => \$opt_version,
 "S|socket=s"        => \$opt_socket,
  “P|port=i”         => \$opt_port
```

把 "e|engine|type=s" => $opt_type 修改为了 "e|engine|type=s" => $opt_engine。因为在上面定义变量时 $opt_engine="MYISAM"，指定了默认的转换是 MyISAM，如果不加 --engine 选项，默认都是转为 MyISAM。当指定 --engine 选项时，应该由 $opt_engine 接受赋值，但原程序错写为使用 $opt_type 接受值，也就是 $opt_engine 依然是默认的值 MyISAM。所以当前面我们指定 --engine=InnoDB，实际上并不起作用，而是按默认的 MyISAM 处理。因为表本身已经是 MyISAM，所以再转为 MyISAM 出现了“已经是 MyISAM”的提示消息。

使用修改过的 mysql_convert_table_format 再次尝试转为 InnoDB，没有再出现提示消息。

```
[root@www bin]# mysql_convert_table_format --host=localhost --user=root --password=111111
      --socket=/tmp/mysql.sock --engine=InnoDB client test
```

查看 test 表的存储引擎，已经是 InnoDB，问题得到了解决。

```
mysql> use client;
Database changed

mysql> show create table test;
| Table | Create Table
| test  | CREATE TABLE 'test' (
  'id' int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8        //InnoDB，转换成功
```

本章总结

- MyISAM 适合在无事务要求，读写并发少的场景使用。
- InnoDB 适合在有事务要求，读写并发多的场景使用。
- MySQL 存储引擎是 MySQL 数据库服务器中的一个组件，负责为数据库执行实际的数据 I/O 操作。
- 修改存储引擎有多种方式，应根据当前数据库的状态使用不同的方式。

本章作业

1．简述 MyISAM 和 InnoDB 的特点及使用场景。

2．创建两个表，分别使用 MyISAM 和 InnoDB 作为存储引擎。使用 begin 命令开启数据库事务分别插入 1 条数据，然后使用 rollback 进行事务回滚，使用 select 查看两种存储引擎的数据库表中数据的变化。

3．用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



随手笔记



第7章

MySQL 主从复制与读写分离

技能目标

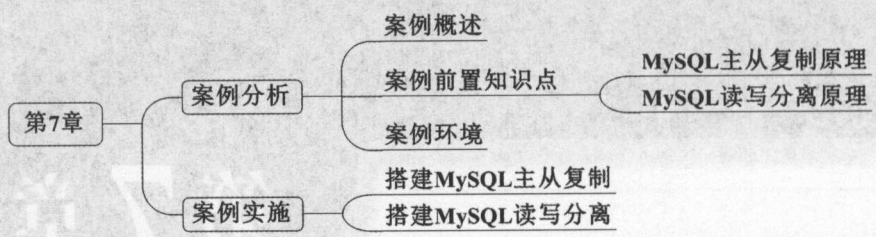
- 熟悉 MySQL 主从复制原理
- 熟悉 MySQL 读写分离原理
- 学会配置 MySQL 主从复制
- 学会配置 MySQL 读写分离

本章导读

在实际的生产环境中，如果对 MySQL 数据库的读和写都在一台数据库服务器中操作，无论是在安全性、高可用性，还是高并发等各个方面都是不能满足实际需求的。本章首先讲解 MySQL 主从复制和 MySQL 读写分离的原理，然后介绍如何配置 MySQL 主从复制和 MySQL 读写分离。

知识服务





7.1 案例分析

7.1.1 案例概述

在实际生产环境中，如果对数据库的读和写都在同一个数据库服务器中操作，无论是在安全性、高可用性，还是高并发等各个方面都是完全不能满足实际需求的，因此，一般来说都是通过主从复制（Master-Slave）的方式来同步数据，再通过读写分离来提升数据库的并发负载能力这样的方案来进行部署与实施。

如图 7.1 所示，一台主 MySQL 带两台从 MySQL 进行了数据复制，前端应用在进行数据库写操作时，对主设备进行操作，在进行数据库读操作时，对两台从设备进行操作，这样大量减轻了对主设备的压力。

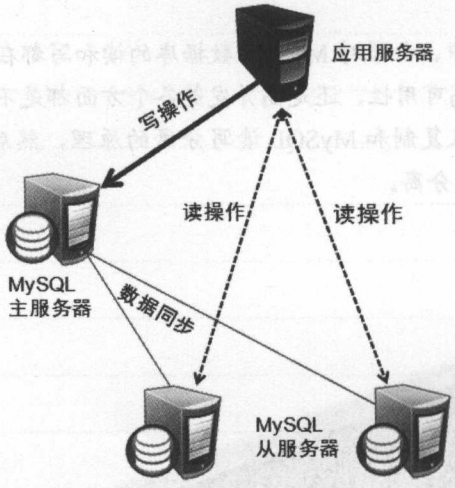


图 7.1 MySQL 主从复制与读写分离

7.1.2 案例前置知识点

1. MySQL 主从复制原理

MySQL 的主从复制和 MySQL 的读写分离两者有着紧密联系，首先要部署主从复

制，只有主从复制完成了，才能在此基础上进行数据的读写分离。

（1）MySQL 支持的复制类型

1）基于语句的复制。在主服务器上执行的 SQL 语句，在从服务器上执行同样的语句。MySQL 默认采用基于语句的复制，效率比较高。

2）基于行的复制。把改变的内容复制过去，而不是把命令在从服务器上执行一遍。

3）混合类型的复制。默认采用基于语句的复制，一旦发现基于语句无法精确复制时，就会采用基于行的复制。

（2）复制的工作过程

MySQL 复制的工作过程如图 7.2 所示。

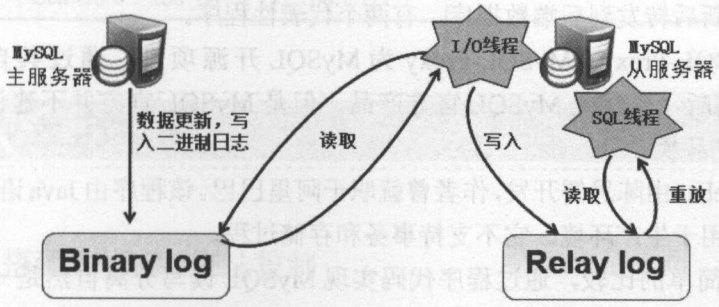


图 7.2　MySQL 复制的工作过程

1）在每个事务更新数据完成之前，Master 在二进制日志记录这些改变。写入二进制日志完成后，Master 通知存储引擎提交事务。

2）Slave 将 Master 的 Binary log 复制到其中继日志。首先，Slave 开始一个工作线程——I/O 线程，I/O 线程在 Master 上打开一个普通的连接，然后开始 Binlog dump process。Binlog dump process 从 Master 的二进制日志中读取事件，如果已经跟上 Master，它会睡眠并等待 Master 产生新的事件。I/O 线程将这些事件写入中继日志。

3）SQL slave thread（SQL 从线程）处理该过程的最后一步。SQL 线程从中继日志读取事件，并重放其中的事件而更新 Slave 的数据，使其与 Master 中的数据一致。只要该线程与 I/O 线程保持一致，中继日志通常会位于 OS 的缓存中，所以中继日志的开销很小。

复制过程有一个很重要的限制，即复制在 Slave 上是串行化的，也就是说 Master 上的并行更新操作不能在 Slave 上并行操作。

2. MySQL 读写分离原理

简单来说，读写分离（见图 7.3）就是只在主服务器上写，只在从服务器上读。基本原理是让主数据库处理事务性查询，而从数据库处理 select 查询。数据库复制被用来把事务性查询导致的变更同步到群集中的从数据库。

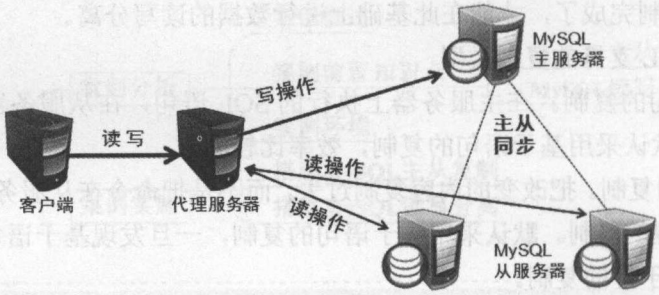


图 7.3　MySQL 读写分离过程

基于中间代理层实现：代理一般位于客户端和服务器之间，代理服务器接到客户端请求通过判断后转发到后端数据库，有两个代表性程序。

（1）MySQL-Proxy。MySQL-Proxy 为 MySQL 开源项目，通过其自带的 lua 脚本进行 SQL 判断，虽然是 MySQL 官方产品，但是 MySQL 官方并不建议将 MySQL-Proxy 用到生产环境。

（2）Amoeba。由陈思儒开发，作者曾就职于阿里巴巴。该程序由 Java 语言进行开发，阿里巴巴将其用于生产环境。它不支持事务和存储过程。

经过上述简单的比较，通过程序代码实现 MySQL 读写分离自然是一个不错的选择，但是并不是所有的应用都适合在程序代码中实现读写分离，像一些大型复杂的 Java 应用，如果在程序代码中实现读写分离对代码改动就较大。所以，像这种应用一般会考虑使用代理层来实现。

7.1.3　案例环境

本案例环境使用五台服务器模拟搭建，具体的拓扑如图 7.4 所示。

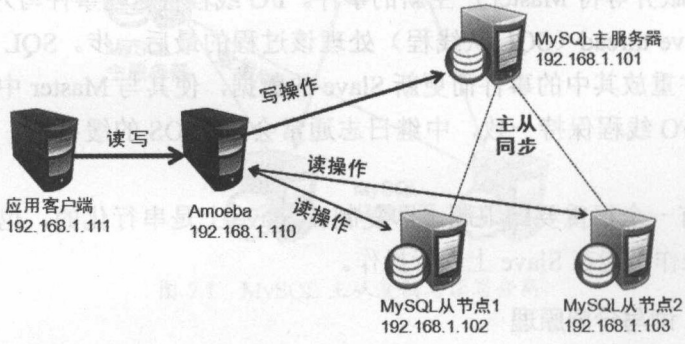


图 7.4　群集拓扑

案例环境如表 7-1 所示。

表 7-1　案例环境

主机	操作系统	IP 地址	主要软件
Master	CentOS 6.5 x86_64	192.168.1.101	cmake-2.8.6.tar.gz mysql-5.5.22.tar.gz
Slave1	CentOS 6.5 x86_64	192.168.1.102	cmake-2.8.6.tar.gz mysql-5.5.22.tar.gz
Slave2	CentOS 6.5 x86_64	192.168.1.103	cmake-2.8.6.tar.gz mysql-5.5.22.tar.gz
Amoeba	CentOS 6.5 x86_64	192.168.1.110	amoeba-mysql-binary-2.2.0.tar.gz jdk-6u14-linux-x64.bin
客户端	CentOS 6.5 x86_64	192.168.1.111	

7.2　案例实施

7.2.1　搭建 MySQL 主从复制

（1）建立时间同步环境，在主节点上搭建时间同步服务器。

1）安装 NTP。

```
[root@localhost ~]# yum -y install ntp
```

2）配置 NTP。

```
[root@localhost ~]# vim /etc/ntp.conf      # 添加如下两行
  server 127.127.1.0
  fudge 127.127.1.0 stratum 8
```

3）重启服务。

```
[root@localhost ~]# service ntpd restart
```

（2）在从节点上进行时间同步。

```
[root@localhost ~]# yum -y install ntpdate
[root@localhost ~]# /usr/sbin/ntpdate 192.168.1.101
```

（3）在每台服务器上关闭 iptables 或者指定端口进行开放。

```
[root@localhost ~]# service iptables stop
[root@localhost ~]# chkconfig iptables off
```

（4）安装 MySQL 数据库。在 Master、Slave1、Slave2 上安装。

1）编译安装 MySQL。

```
[root@localhost ~]# yum -y install ncurses-devel
```

```
[root@localhost ~]# tar zxvf cmake-2.8.6.tar.gz
[root@localhost ~]# cd cmake-2.8.6
[root@localhost cmake-2.8.6]# ./configure
[root@localhost cmake-2.8.6]# gmake
[root@localhost cmake-2.8.6]# gmake install

[root@localhost ~]# tar zxvf mysql-5.5.22.tar.gz
[root@localhost ~]# cd mysql-5.5.22
[root@localhost mysql-5.5.22]# cmake -DCMAKE_INSTALL_PREFIX=/usr/local/mysql
    -DDEFAULT_CHARSET=utf8  -DDEFAULT_COLLATION=utf8_general_ci -DWITH_
    EXTRA_ CHARSETS=all -DSYSCONFDIR=/etc
[root@localhost mysql-5.5.22]# make && make install
```

2）优化调整。

```
[root@localhost mysql-5.5.22]# cp support-files/my-medium.cnf /etc/my.cnf
[root@localhost mysql-5.5.22]# cp support-files/mysql.server /etc/rc.d/init.d /mysqld
[root@localhost mysql-5.5.22]# chmod +x /etc/rc.d/init.d/mysqld
[root@localhost mysql-5.5.22]# chkconfig --add mysqld
[root@localhost mysql-5.5.22]# echo "PATH=$PATH:/usr/local/mysql/bin" >> /etc profile
[root@localhost mysql-5.5.22]# . /etc/profile
```

3）初始化数据库。

```
[root@localhost mysql-5.5.22]# groupadd mysql
[root@localhost mysql-5.5.22]# useradd -M -s /sbin/nologin mysql -g mysql
[root@localhost mysql-5.5.22]# chown -R mysql:mysql /usr/local/mysql
[root@localhost mysql-5.5.22]# /usr/local/mysql/scripts/mysql_install_db
    --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data --user=mysql
```

4）启动 MySQL 服务。

```
[root@localhost ~]# service mysqld start
[root@localhost ~]# chkconfig mysqld on
[root@localhost ~]# mysqladmin -u root password 'pwd123'
                                    // 为 root 用户设置密码
```

（5）配置 MySQL Master 主服务器。

1）在 /etc/my.cnf 中修改或者增加以下内容。

```
[root@localhost ~]# vim /etc/my.cnf
server-id       = 11                    // 修改
log-bin=master-bin                      // 修改
log-slave-updates=true                  // 增加
```

2）重启 MySQL 服务。

```
[root@localhost ~]# service mysqld restart
```

3）登录 MySQL 程序，给从服务器授权。

```
[root@localhost ~]# mysql -u root -p
mysql>GRANT REPLICATION SLAVE ON *.* TO 'myslave'@'192.168.1.%' IDENTIFIED
      BY '123456';
mysql>FLUSH PRIVILEGES;
mysql>show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| master-bin.000001 |      339 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

其中 File 列显示日志名，Position 列显示偏移量，这两个值在后面配置从服务器的时候会用到。Slave 应从该点在 Master 上进行新的更新。

（6）配置从服务器。

1）在 /etc/my.cnf 中修改或者增加以下内容。

```
[root@localhost ~]# vim /etc/my.cnf
server-id       = 22                          // 修改
relay-log=relay-log-bin                       // 增加
relay-log-index=slave-relay-bin.index         // 增加
```

这里要注意 server-id 不能与主服务器相同。

2）重启 MySQL 服务。

```
[root@localhost ~]# service mysqld restart
```

3）登录 MySQL，配置同步。

按主服务器结果更改下面命令中 master_log_file 和 master_log_pos 的参数。

```
[root@localhost ~]# mysql -u root -p
mysql>change master to master_host='192.168.1.101',master_user='myslave',
master_password='123456',master_log_file='master-bin.000001',master_log_pos=339;
```

4）启动同步。

```
mysql>start slave;
```

5）查看 Slave 状态，确保以下两个值为 YES。

```
mysql> show slave status\G;
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.1.101
                Master_User: myslave
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: master-bin.000001
```

```
        Read_Master_Log_Pos: 339
            Relay_Log_File: relay-log-bin.000002
             Relay_Log_Pos: 254
      Relay_Master_Log_File: master-bin.000001
          Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
           Replicate_Do_DB:
       Replicate_Ignore_DB:
```

（7）验证主从复制效果。

1）在主、从服务器上登录 MySQL。

```
[root@localhost ~]# mysql -u root -p
mysql> show databases;
```

两台数据库执行结果应该相同。

2）在主服务器上新建数据库 db_test。

```
mysql> create database db_test;
```

3）在主、从服务器上分别查看数据库，显示数据库相同，则主从复制成功。

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| db_test            |
| mysql              |
| performance_schema |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

7.2.2 搭建 MySQL 读写分离

Amoeba（变形虫）项目开源框架于 2008 年发布了一款 Amoeba for MySQL 软件。这个软件致力于 MySQL 的分布式数据库前端代理层，它主要为应用层访问 MySQL 时充当 SQL 路由，并具有负载均衡、高可用性、SQL 过滤、读写分离、可路由相关到目标数据库、可并发请求多台数据库。通过 Amoeba 能够完成多数据源的高可用、负载均衡、数据切片的功能，目前 Amoeba 已在很多企业的生产线上使用，其版本可在官网进行下载。

（1）在主机 Amoeba 上安装 Java 环境。

因为 Amoeba 是基于 jdk1.5 开发的，所以官方推荐使用 jdk1.5 或 1.6 版本，高版本不建议使用。

```
[root@localhost ~]#chmod +x /usr/local/jdk-6u14-linux-x64.bin
[root@localhost ~]#./jdk-6u14-linux-x64.bin     // 根据提示按 Enter 键完成即可
[root@localhost ~]# mv jdk1.6.0_14/ /usr/local/jdk1.6
[root@localhost ~]#vim /etc/profile
```

增加以下配置：

```
export JAVA_HOME=/usr/local/jdk1.6
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
export PATH=$JAVA_HOME/lib:$JAVA_HOME/jre/bin:$PATH:$HOME/bin
export AMOEBA_HOME=/usr/local/amoeba/
export PATH=$PATH:$AMOEBA_HOME/bin

[root@localhost ~]#source /etc/profile
[root@localhost ~]# java -version
java version "1.6.0_14"
Java(TM) SE Runtime Environment (build 1.6.0_14-b08)
Java HotSpot(TM) 64-Bit Server VM (build 14.0-b16, mixed mode)
```

Java 环境已配置成功。

（2）安装并配置 Amoeba 软件。

```
[root@localhost ~]#mkdir /usr/local/amoeba
[root@localhost ~]#tar zxf /root/ amoeba-mysql-binary-2.2.0.tar.gz -C /usr/local /amoeba
[root@node5 ~]#chmod -R 755 /usr/local/amoeba/
[root@localhost ~]#/usr/local/amoeba/bin/amoeba
amoeba start|stop    // 显示此内容说明 Amoeba 安装成功
```

（3）配置 Amoeba 读写分离，两个 Slave 读负载均衡。

1）Master、Slave1、Slave2 中开放权限给 Amoeba 访问。

```
grant all on *.* to test@'192.168.1.%' identified by '123.com';
```

2）编辑 amoeba.xml 配置文件。

```
[root@localhost amoeba]# vim conf/amoeba.xml
.....  // 省略部分内容
```

以下修改后的内容为带加粗的部分。

```
<property name="authenticator">
    <bean class="com.meidusa.amoeba.mysql.server.MysqlClientAuthenticator">
        <Property name="user">amoeba</property>
        <property name="password">123456</property>
        <property name="filter">
            <bean class="com.meidusa.amoeba.server.IPAccessController">
                <property name="ipFile">${amoeba.home}/conf/access_ list. conf
                </property>
            </bean>
        </property>
```

```
    </bean>
</property>
..... // 省略部分内容
```

以下修改后的内容为带加粗的部分，注意删除注释。

```
<queryRouter class="com.meidusa.amoeba.mysql.parser.MysqlQueryrouter">
    <property name="ruleLoader">
        <bean class="com.meidusa.amoeba.route.TableRuleFileLoader">
            <property name="rulfFile">${amoeba.home}/conf/rule.xml </property>
            <property name="functionFile">${amoeba.home}/conf/ruleFunctionMap.xml
            </property>
        </bean>
    </property>
    <property name="sqlFunctionFile">${amoeba.home}/conf/functionMap.xml </property>
    <property name="LRUMapSize">1500</property>
    <property name="defaultPool">master</property>

    <property name="writePool">master</property>
    <property name="readPool">slaves</property>

    <property name="needParse">true</property>
</queryRouter>
```

3）编辑 dbServers.xml 配置文件。

```
[root@localhost amoeba]# vim conf/dbServers.xml
..... // 省略部分内容
```

以下修改后的内容为带加粗的部分。

```
    <!-- mysql user -->
    <property name="user">test</property>

    <property name="password">123.com</property>
</factoryConfig>
..... // 省略部分内容
```

以下修改后的内容为带加粗的部分。

```
<dbServer name="master"  parent="abstractServer">
    <factoryConfig>
        <!-- mysql ip-->
        <property name="ipAddress">192.168.1.101</property>
    </factoryConfig>
</dbServer>

<dbServer name="slave1"  parent="abstractServer">
    <factoryConfig>
        <!-- mysql ip -->
```

```
        <property name="ipAddress">192.168.1.102</property>
    </factoryConfig>
</dbServer>

<dbServer name="slave2"  parent="abstractServer">
    <factoryConfig>
        <!-- mysql ip -->
        <property name="ipAddress">192.168.1.103</property>
    </factoryConfig>
</dbServer>
<dbServer name="slaves" virtual="true">
    <poolConfig class="com.meidusa.amoeba.server.MultipleServerPool">
        <!-- Load balancing strategy: 1=ROUNDROBIN, 2=WEIGHTBASED, 3=HA-->
        <property name="loadbalance">1</property>

        <!-- Separated by commas,such as: server1,server2,server1 -->
        <property name="poolNames">slave1,slave2</property>
    </poolConfig>
</dbServer>
```

4）配置无误后，可以启动 Amoeba 软件，其默认端口为 tcp 8066。

```
[root@localhost amoeba]# bin/amoeba start&
[root@localhost amoeba]# netstat -anpt | grep java
tcp   0   0 ::ffff:127.0.0.1:7776      :::*              LISTEN      9547/java
tcp   0   0 :::8066                    :::*              LISTEN      9547/java
```

（4）测试。

1）在 Client 主机上。

```
[root@localhost ~]# yum -y install mysql
```

可以通过代理访问 MySQL：

```
[root@localhost ~]# mysql -u amoeba -p123456 -h 192.168.1.110 -P8066
..... // 省略部分内容
mysql>
```

2）在 Master 上创建一个表，同步到各从服务器上，然后关闭各从服务器的 Slave 功能，再插入区别语句。

```
mysql> use db_test;
Database changed
mysql> create table zang (id int(10),name varchar(10),address varchar(20));
Query OK, 0 rows affected (0.16 sec)
```

分别在两台从服务器上关闭 slave 功能：

```
mysql> stop slave;
```

然后在主服务器上插入区别语句：

```
mysql> insert into zang values('1','zhang','this_is_master');
Query OK,1 row affected (0.06 sec)
```

3）从服务器上同步了表，手动插入其他内容。

```
slave1:
mysql> use db_test;
Database changed
mysql> insert into zang values('2','zhang','this_is_slave1');
Query OK,1 row affected (0.02 sec)

slave2:
mysql> use db_test;
Database changed
mysql> insert into zang values('3','zhang','this_is_slave2');
Query OK,1 row affected (0.02 sec)
```

4）测试读操作。

在 Client 主机上第一次查询的结果如下：

```
mysql> select * from zang;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    2 | zhang | this_is_slave1 |
+------+-------+---------------+
1 row in set (0.01 sec)
```

第二次查询的结果如下：

```
mysql> select * from zang;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    3 | zhang | this_is_slave2 |
+------+-------+---------------+
1 row in set (0.01 sec)
```

第三次查询的结果如下：

```
mysql> select * from zang;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    2 | zhang | this_is_slave1 |
+------+-------+---------------+
1 row in set (0.01 sec)
```

5）测试写操作。

在 Client 主机上插入一条语句：

```
mysql> insert into zang values('4','zhang','write_test');
Query OK,1 row affected (0.08 sec)
```

但在 Client 上查询不到，最终只有在 Master 上才能查看到这条语句内容，说明写操作在 Master 服务器上。

```
mysql> select * from zang;
+------+-------+---------------+
| id   | name  | address       |
+------+-------+---------------+
|    1 | zhang | this_is_master |
|    4 | zhang | write_test    |
+------+-------+---------------+
2 row in set (0.01 sec)
```

由此验证，已经实现了 MySQL 读写分离，目前所有的写操作全部在 Master 主服务器上，用来避免数据的不同步；所有的读操作都分摊给了 Slave 从服务器，用来分担数据库的压力。

本章总结

- MySQL 的主从复制和 MySQL 的读写分离两者有着紧密联系，首先要部署主从复制，只有主从复制完成了，才能在此基础上进行数据的读写分离。
- MySQL 支持的复制类型有三种：基于语句的复制、基于行的复制和混合类型的复制。
- 通过程序代码实现 MySQL 读写分离是一个不错的选择，但是并不是所有的应用都适合在程序代码中实现读写分离，像一些大型复杂的 Java 应用，一般会考虑使用代理层来实现。

本章作业

1. 画图描述 MySQL 主从复制的原理。
2. 画图描述 MySQL 读写分离的原理。
3. 描述 MySQL 读写分离的测试方法及步骤。
4. 基于 CentOS 7.3 使用 MySQL 5.7 搭建主从复制和读写分离。

随手笔记



第8章

MySQL 高可用群集

技能目标

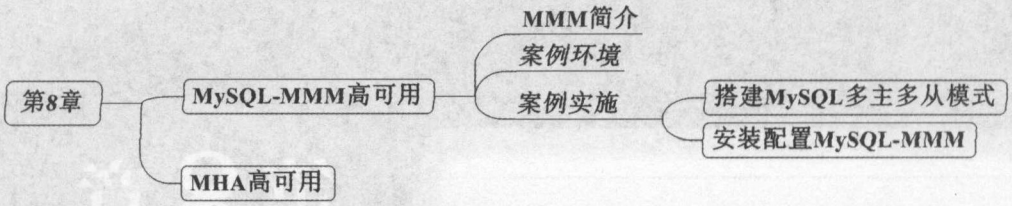
- 熟悉 MMM 高可用架构
- 熟悉 MHA 高可用架构
- 学会配置 MMM 高可用
- 学会配置 MHA 高可用

本章导读

在实际的生产环境中，当 MySQL 主从复制和读写分离不能满足实际需求时，就需要考虑 MySQL 高可用群集了。本章将介绍 MySQL-MMM 实现 MySQL 高可用以及 MHA 高可用。

知识服务





8.1 MySQL-MMM 高可用

8.1.1 MMM 简介

MMM（Master-Master replication manager for MySQL，MySQL 主主复制管理器）是一套支持双主故障切换和双主日常管理的脚本程序。MMM 使用 Perl 语言开发，主要用来监控和管理 MySQL Master-Master（双主）复制，虽然叫做双主复制，但是业务上同一时刻只允许对一个主进行写入，另一台备选主上提供部分读服务，以加速在主主切换时备选主的预热，可以说 MMM 这套脚本程序一方面实现了故障切换的功能，另一方面其内部附加的工具脚本也可以实现多个 Slave 的 read 负载均衡。

MMM 提供了自动和手动两种方式移除一组服务器中复制延迟较高的服务器的虚拟 ip，同时它还可以备份数据，实现两节点之间的数据同步等。由于 MMM 无法完全保证数据的一致性，所以 MMM 适用于对数据的一致性要求不是很高，但是又想最大程度地保证业务可用性的场景。对于那些对数据的一致性要求很高的业务，非常不建议采用 MMM 这种高可用架构。

MMM 是一套灵活的脚本程序，基于 perl 实现，用来对 mysql replication 进行监控和故障迁移，并能管理 MySQL Master-Master 复制的配置，如图 8.1 所示。

关于 MMM 高可用架构的说明如下：

- mmm_mon：监控进程，负责所有的监控工作，决定和处理所有节点角色活动。此脚本需要在监管机上运行。
- mmm_agent：运行在每个 MySQL 服务器上的代理进程，完成监控的探针工作和执行简单的远端服务设置。此脚本需要在被监管机上运行。
- mmm_control：一个简单的脚本，提供管理 mmm_mond 进程的命令。
- mysql-mmm 的监管端会提供多个虚拟 IP（VIP），包括一个可写 VIP，多个可读 VIP，通过监管的管理，这些 IP 会绑定在可用 MySQL 之上，当某一台 MySQL 宕机时，监管会将 VIP 迁移至其他 MySQL。

在整个监管过程中，需要在 MySQL 中添加相关授权用户，以便让 MySQL 可以支持监理机的维护。授权的用户包括一个 mmm_monitor 用户和一个 mmm_agent 用户，如果想使用 MMM 的备份工具则还要添加一个 mmm_tools 用户。

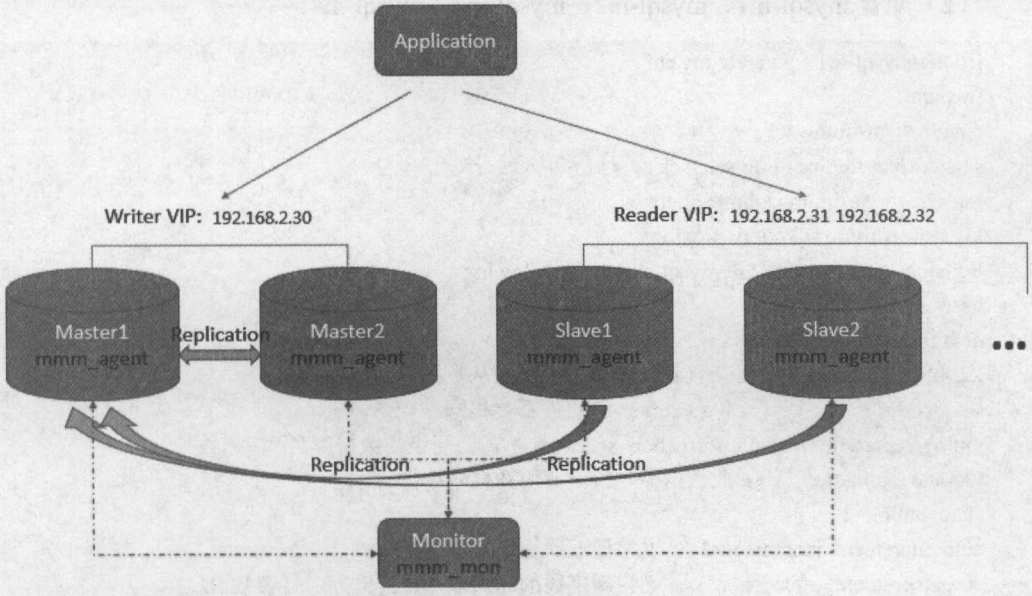


图 8.1　MMM 高可用架构

8.1.2　案例环境

本案例环境使用五台服务器模拟搭建，案例环境如表 8-1 所示。

表 8-1　案例环境

主机	操作系统	IP 地址	主要软件
mysql-m1 主服务器	Centos 6.5 x86_64	192.168.2.21	MySQL-MMM 2.2.1
mysql-m2 主服务器	Centos 6.5 x86_64	192.168.2.22	MySQL-MMM 2.2.1
mysql-monitor	Centos 6.5 x86_64	192.168.2.24	MySQL-MMM 2.2.1
mysql-m3 从服务器	Centos 6.5 x86_64	192.168.2.25	MySQL-MMM 2.2.1
mysql-m4 从服务器	Centos 6.5 x86_64	192.168.2.26	MySQL-MMM 2.2.1

8.1.3　案例实施

1. 搭建 MySQL 多主多从模式

（1）安装 MySQL。

```
[root@mysql-m1 ~]#yum -y install mysqlmysql-server mysql-devel
[root@mysql-m1 ~]#mysqladmin-u root password 'yunjisuan'
```

mysql-m2、mysql-m3、mysql-m4 主机也同样安装 MySQL。

（2）配置 mysql-m1、mysql-m2、mysql-m3、mysql-m4。

```
[root@mysql-m1 ~]#vi /etc/my.cnf
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
log-error=/var/lib/mysql/mysql.err
log = /var/lib/mysql/query_log.log
log-slow-queries = /var/lib/mysql/slow_query_log.log
user=mysql
default-character-set=utf8
log-bin=mysql-bin       # 开启 binlog 日志用于主从数据复制
server-id=1             # 每台 server-id 的值不要相同
binlog-ignore-db=mysql,information_schema# 不需要同步的数据库名称
log-slave-updates       # 此数据库宕机，备用数据库接管
sync_binlog=1
auto_increment_increment=2      # 字段一次递增多少
auto_increment_offset=1         # 自增字段的起始值：1, 3, 5, 7, ... 等奇数 ID
[client]
default-character-set=utf8
[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

注释：

每台 MySQL 主机的 server-id 不能相同，其他配置文件参数相同即可！

```
[root@mysql-m1 ~]#/etc/init.d/mysqld restart
```

（3）配置 mysql-m1、mysql-m2 主主模式。

1）先查看 log bin 日志和 pos 值的位置。

```
[root@mysql-m1 ~]#mysql-u root -pyunjisuan
mysql> show master status;
+------------------+----------+--------------+--------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB         |
+------------------+----------+--------------+--------------------------+
| mysql-bin.000005 |      106 |              | mysql，information_schema |
+------------------+----------+--------------+----------------------
1 row in set (0.00 sec)
[root@mysql-m2 ~]#mysql -u root -pyunjisuan
mysql> show master status;
+------------------+----------+--------------+--------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB         |
+------------------+----------+--------------+--------------------------+
| mysql-bin.000005 |      106 |              | mysql，information_schema |
```

```
+-----------------+----------+-------------+----------------------
1 row in set (0.00 sec)
```

2）mysql-m1、mysql-m2 互相提升访问权限。

```
mysql-m1:
mysql> grant replication slave on *.* to 'replication'@'192.168.2.22' identified by 'yunjisuan';
mysql> flush privileges;
mysql-m2:
mysql> grant replication slave on *.* to 'replication'@'192.168.2.21' identified by 'yunjisuan';
mysql> flush privileges;
mysql>change master to master_host='192.168.2.21', master_user='replication',master_
    password='yunjisuan',master_log_file='mysql-bin.000005',master_log_pos=106;
mysql-m1:
mysql>change master to master_host='192.168.2.22', master_user='replication',master_
password='yunjisuan',master_log_file='mysql-bin.000005',master_log_pos=106;
```

3）分别查看 mysql-m1、mysql-m2 服务器的主从状态。

```
mysql>start slave;
mysql>show slave status\G;
```

结果如图 8.2 所示。

```
mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.2.21
                  Master_User: replication
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000005
          Read_Master_Log_Pos: 106
               Relay_Log_File: mysqld-relay-bin.000012
                Relay_Log_Pos: 251
        Relay_Master_Log_File: mysql-bin.000005
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

图 8.2　查看服务器主从状态

主主同步配置完毕，查看同步状态 Slave_IO 和 Slave_SQL 为 YES，说明主主同步成功。

4）测试主主同步，在 mysql-m2 新建一个库 kgc。

```
mysql> create database kgc;
Query OK, 1 row affected (0.06 sec)
```

回到 mysql-m1 数据库服务器，查看刚才创建的库 kgc，同步已经完成。

```
mysql> show databases;
+--------------------+
| Database           |
```

```
+--------------------+
| information_schema |
| kgc                |
| mysql              |
| test               |
+--------------------+
5 rows in set (0.00 sec)
```

（4）配置 mysql-m3 和 mysql-m4 作为 mysql-m1 的从库。

1）先查看 mysql-m1 master 的状态值。

```
mysql> show master status;
+------------------+----------+--------------+--------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB         |
+------------------+----------+--------------+--------------------------+
| mysql-bin.000005 |      192 |              | mysql, information_schema |
+------------------+----------+--------------+--------------------------
1 row in set (0.00 sec)
```

2）在 mysql-m3 和 mysql-m4 分别执行。

```
mysql>changemaster to master_host='192.168.2.21', master_user='replication',master_
password='yunjisuan',master_log_file='mysql-bin.000005',master_log_pos=192;
```

3）分别查看 mysql-m3 和 mysql-m4 服务器的主从状态，如图 8.3 所示。

```
mysql> show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.2.21
                  Master_User: replication
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000005
          Read_Master_Log_Pos: 192
               Relay_Log_File: mysqld-relay-bin.000006
                Relay_Log_Pos: 337
        Relay_Master_Log_File: mysql-bin.000005
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

图 8.3　查看服务器主从状态

在 mysql-m3 和 mysql-m4 可以看到主从配置复制成功。

2. 安装配置 MySQL-MMM

（1）安装 MMM。

CentOS 默认没有 mysql-mmm 软件包，官方推荐使用 epel 源，五台主机都要安装 epel 源和 MMM。

```
[root@monitor ~]#wget http://mirrors.yun-idc.com/epel/6/x86_64/epel-release-6-8.noarch.rpm
[root@monitor ~]#rpm -ivh epel-release-6-8.noarch.rpm
[root@monitor ~]#yum -y install mysql-mmm*
```

（2）在 mysql-m1、mysql-m2、mysql-m3、mysql-m4 节点授权 monitor 访问。

```
[root@db ~]# mysql -u root -pyunjisuan
mysql> grant replication client on *.* to 'mmm_monitor'@'192.168.2.%' identified by 'monitor';
mysql>grantsuper,replicationclient, processon*.*to 'mmm_agent'@'192.168.2.%' identifiedby'agent';
```

（3）修改 /etc/mysql-mmm/mmm_common.conf 配置文件，系统中所有主机的该配置文件内容都是一样的，包括监控主机 mysql-monitor。

```
[root@mysql-m4 ~]# vi/etc/mysql-mmm/mmm_common.conf
active_master_role      writer
<host default>
cluster_interface       eth0
pid_path                /var/run/mysql-mmm/mmm_agentd.pid
bin_path                /usr/libexec/mysql-mmm/
replication_user        replication
replication_passwordyunjisuan
agent_usermmm_agent
agent_password          agent
</host>
<host db1>
ip      192.168.2.21
mode    master
peer    db2
</host>
<host db2>
ip      192.168.2.22
mode    master
peer    db1
</host>
<host db3>
ip      192.168.2.25
mode    slave
</host>
<host db4>
ip      192.168.2.26
mode    slave
</host>
<role writer>
hosts   db1, db2
ips     192.168.2.30
    mode    exclusive       # 只有一个 host 可以进行写操作模式
</role>
<role reader>
hosts   db3, db4
ips     192.168.2.31,192.168.2.32
    mode    balanced        # 多个 slave 主机可以进行读操作模式
</role>
```

（4）在数据库主机上需要编辑 /etc/mysql-mmm/mmm_agent.conf 文件，根据不同的主机修改为不同的值。

```
[root@mysql-m1 ~]#vi/etc/mysql-mmm/mmm_agent.conf
includemmm_common.conf
this db1        # 分别修改为 db1、db2、db3 和 db4
```

（5）在监控主机上需要编辑 /etc/mysql-mmm/mmm_mon.conf 文件。

```
[root@monitor ~]#vi /etc/mysql-mmm/mmm_mon.conf
include mmm_common.conf
<monitor>
ip              127.0.0.1
pid_path          /var/run/mysql-mmm/mmm_mond.pid
bin_path          /usr/libexec/mysql-mmm
status_path        /var/lib/mysql-mmm/mmm_mond.status
ping_ips192.168.2.21,192.168.2.22,192.168.2.25,192.168.2.26  # 监控服务器的 ip
auto_set_online    10
   # The kill_host_bin does not exist by default, though the monitor will
   # throw a warning about it missing.  See the section 5.10 "Kill Host
# Functionality" in the PDF documentation.
   #
   # kill_host_bin     /usr/libexec/mysql-mmm/monitor/kill_host
   #
</monitor>
<host default>
monitor_usermmm_monitor#mmm_monitor 用户名
monitor_password    monitor#mmm_monitor 密码
</host>
debug 0
```

（6）启动代理和监控。

```
[root@mysql-m1 ~]# /etc/init.d/mysql-mmm-aget start
[root@mysql-m2 ~]# /etc/init.d/mysql-mmm-aget start
[root@mysql-m3 ~]# /etc/init.d/mysql-mmm-aget start
[root@mysql-m4 ~]# /etc/init.d/mysql-mmm-aget start
[root@monitor ~]# /etc/init.d/mysql-mmm-monitor start
```

（7）测试群集。

```
[root@monitor mysql-mmm]# mmm_control show
db1(192.168.2.21) master/ONLINE. Roles: writer(192.168.2.30)
db2(192.168.2.22) master/HARD_OFFLINE. Roles:
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.31)
db4(192.168.2.26) slave/ONLINE. Roles: reader(192.168.2.32)
```

由此看来，主服务器 mysql-m1 是一个对外写入的角色，但不真正提供写，要想实现读写分离还需要结合 Amoeba。后面的虚拟 IP 是真正来访问 MySQL 数据库的。

（8）故障转移切换。

1）停掉主 db1 数据库，等待几秒后，可以看到数据库 db1 处于 HARD_OFFLINE（离线状态），检测不到数据库的存在。

```
[root@monitor mysql-mmm]# mmm_control show
db1(192.168.2.21) master/ONLINE. Roles:
db2(192.168.2.22) master/ONLINE. Roles: writer(192.168.2.30)
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.31)
db4(192.168.2.26) slave/ONLINE. Roles: reader(192.168.2.32)
```

宕掉 MySQL 数据库，虚拟 IP 会全部在另一台正常数据库上。

2）模拟主主故障插入数据测试。

```
[root@monitor ~]# mmm_control show
db1(192.168.2.21) master/ONLINE. Roles: writer(192.168.2.30)  // 正常状态下
db2(192.168.2.22) master/HARD_OFFLINE. Roles:
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.32)
db4(192.168.2.26) slave/ONLINE. Roles: reader(192.168.2.31)
[root@mysql-m1 ~]#/etc/init.d/mysqld stop
```

用虚拟 IP 连接数据库，进行数据插入测试，下面可以看到，当 mysql-m1 数据停止后，在进入某一个库时，出现这样一个现象，最后连续几次才能成功。这说明虚拟 IP 已经成功地漂移到 mysql-m2 数据库上，并为用户提供正常的写入操作。整个切换过程用时 10 秒左右。

```
[root@monitor ~]#mysql -u root -h 192.168.2.30 -pyunjisuan.com
mysql> use kgc;
No connection. Trying to reconnect...
Connection id:    4439
Current database: *** NONE ***   // 有 10 ～ 15 秒的延迟

Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> use kgc;
Database changed
mysql> insert into t1 values('yujisuan');   // 插入一条数据
Query OK, 1 row affected, 1 warning (0.05 sec)
```

接下来回到 mysql-m2 数据库服务器上进行查看，数据是否在 mysql-m2 数据库上。

```
[root@mysql-m2 ~]# mysql -u root -pyunjisuan
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5275
Server version: 5.1.73-log Source distribution
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
```

```
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help.Type '\c' to clear the current input statement.
mysql> use kgc;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql>
mysql> select * from t1;
+------------+
| name       |
+------------+
| yunjisuan  |
+------------+
2 rows in set (0.00 sec)
```

可以看出刚才插入的那条数据在 mysql-m2 数据库上。

启动主 db1 数据库后，可以看到数据库 db1 处于 AWAITING_RECOVER（恢复状态），几秒后将恢复在线状态。

```
[root@monitor ~]# mmm_control show
db1(192.168.2.21) master/HARD_OFFLINE. Roles:    //mysql-m1 数据库关闭的状态
db2(192.168.2.22) master/ONLINE. Roles: writer(192.168.2.30)
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.32)
db4(192.168.2.26) slave/ONLINE. Roles: reader(192.168.2.31)
```

下面启动 mysql-m1 数据库。

```
[root@monitor ~]# mmm_control show
db1(192.168.2.21) master/AWAITING_RECOVERY. Roles:    //mysql-m1 数据库正在恢复过程
db2(192.168.2.22) master/ONLINE. Roles: writer(192.168.2.30)
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.32)
db4(192.168.2.26) slave/ONLINE. Roles: reader(192.168.2.31)
```

等待几秒钟后查看。

```
[root@monitor ~]# mmm_control show
db1(192.168.2.21) master/ONLINE. Roles:      //mysql-m1 已恢复在线状态
db2(192.168.2.22) master/ONLINE. Roles: writer(192.168.2.30)
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.32)
db4(192.168.2.26) slave/ONLINE. Roles: reader(192.168.2.31)
```

这时我们可以进入 mysql-m1 数据库查看主主同步否正常，并查看数据是否同步。

```
[root@mysql-m1 ~]# mysql -u root -pyunjisuan
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 224
Server version: 5.1.71-log Source distribution
```

```
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help.Type '\c' to clear the current input statement.

mysql> use kgc;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
mysql> select * from t1;
+------------+
| name       |
+------------+
|yunjisuan   |
+------------+
2 rows in set (0.00 sec)
```

从上面的数据可以看出，当 mysql-m1 数据库启动以后，数据已经正常同步，mysql-mmm 故障切换工作正常。

模拟主从故障也是如此，DOWN 掉 mysql-m4，虚拟 IP 会全部在另一台正常数据库上。

```
[root@monitor ~]# mmm_control show
db1(192.168.2.21) master/ONLINE. Roles: writer(192.168.2.30)
db2(192.168.2.22) master/ONLINE. Roles:
db3(192.168.2.25) slave/ONLINE. Roles: reader(192.168.2.32) reader(192.168.2.31)
db4(192.168.2.26) slave/ONLINE. Roles:
```


8.2　MHA 高可用

MHA（Master High Availability）目前在 MySQL 高可用方面是一个相对成熟的解决方案，它由日本人 youshimaton 开发，是一套优秀的 MySQL 故障切换和主从提升的高可用软件。在 MySQL 故障切换过程中，MHA 能做到 0 ～ 30 秒之内自动完成数据库的故障切换操作，并且在进行故障切换的过程中，MHA 能最大程度上保证数据库的一致性，以达到真正意义上的高可用。

如图 8.4 所示，MHA 由两部分组成：MHA Manager（管理节点）和 MHA Node（数据节点）。MHA Manager 可以部署在一台独立的机器上管理多个 Master-Slave 群集，也可以部署在一台 Slave 上。当 Master 出现故障时，它可以自动将最新数据的 Slave

提升为新的 Master，然后将所有其他的 Slave 重新指向新的 Master。整个故障转移过程对应用程序是完全透明的。

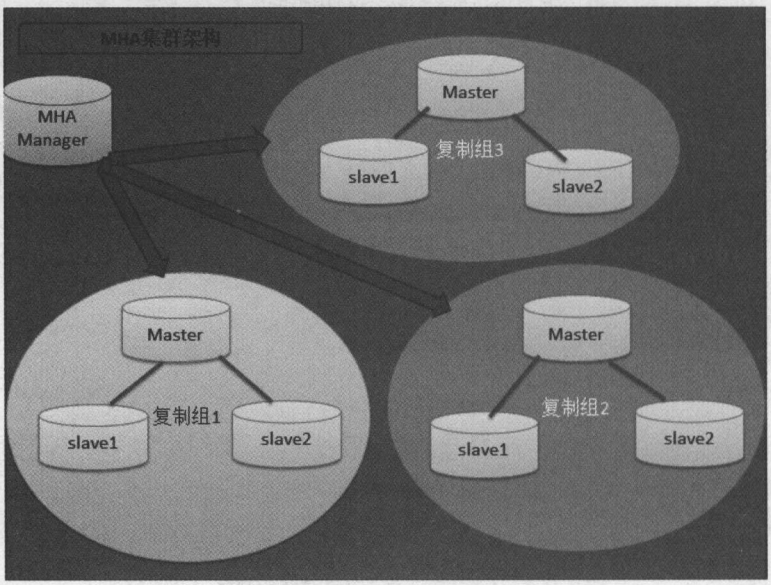


图 8.4　MHA 架构

关于 MHA 的详细介绍请登录课工场 APP 或官网 kgc.cn 观看视频。

本章总结

- MySQL-MMM 适用于对数据的一致性要求不是很高，但是又想最大程度地保证业务可用性的场景。
- MHA 目前在 MySQL 高可用方面是一个相对成熟的解决方案，在进行故障切换的过程中，MHA 能最大程度上保证数据库的一致性，以达到真正意义上的高可用。

本章作业

1. 画图并描述 MHA 的工作原理。
2. 完成 MHA 的搭建及测试。

第9章

MongoDB 基础

技能目标

- 理解 MongoDB 数据库的基本概念
- 学会安装 MongoDB 数据库
- 理解 MongoDB 数据库的存储结构
- 学会 MongoDB 数据库的基本操作
- 学会 MongoDB 数据库的日常维护

本章导读

MongoDB 是一款开源的文档数据库，并且是业内领先的 NoSQL 数据库，用 C++ 编写而成。本课程将介绍 MongoDB 的相关概念，从而能够较为深入地了解如何用它来创建并部署具有高度可扩展性、以性能为导向的数据库。

知识服务





9.1 MongoDB 概述

本节将依次介绍 MongoDB 的基本概念、MongoDB 的特点以及 MongoDB 适用的领域。

9.1.1 MongoDB 简介

MongoDB 是一款跨平台、面向文档的数据库。可以实现高性能、高可用性，并且能够轻松扩展。是一个基于分布式文件存储的开源数据库系统，在高负载的情况下，添加更多的节点，可以保证服务器性能。

MongoDB 也是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。不采用关系模型主要是为了获得更好的扩展性，MongoDB 不再有“行”（row）的概念，其运行方式主要基于两个概念：集合（collection）与文档（document）。

9.1.2 MongoDB 的特点

MongoDB 的特点包括面向集合存储、模式自由、丰富的查询语句和多级索引、复制集机制、易于水平扩展、可插入存储引擎、跨平台多语言支持等。

MongoDB 安装简单，提供了面向文档存储功能，操作起来比较容易。

MongoDB 提供了复制、高可用性和自动分片功能。如果负载增加（需要更多的存储空间和更强的处理能力），它可以分布在计算机网络中的其他节点上，这就是所谓的分片。

Mongo 支持丰富的查询表达式。查询指令使用 JSON 形式的标记，可轻易查询文档中内嵌的对象及数组。

MongoDB 支持各种编程语言：Ruby、Python、Java、C++、PHP、C# 等多种语言。

9.1.3　MongoDB 适用领域

MongoDB 可以为 Web 应用提供可扩展的高性能数据存储解决方案。MongoDB 主要适用领域有网站数据、分布式场景、数据缓存和 JSON 文档格式存储。适合大数据量、高并发、弱事务的互联网应用，其内置的水平扩展机制提供了从几百万到十亿级别的数据处理能力，可以很好地满足 Web2.0 和移动互联网应用数据存储的要求。

9.2　MongoDB 的安装

本节将介绍 MongoDB 的安装、配置、启动、停止和连接数据库等操作。

9.2.1　安装及运行控制

MongoDB 提供了 Linux 平台上的安装包，可以从官方网站 http://www.mongodb.org/downloads 下载。本案例使用 MongoDB 3.2 来介绍 MongoDB 的安装和运行控制。

1. 安装 MongoDB

（1）安装支持软件和 MongoDB

下载完安装包，并解压 tgz，以下演示的是在 64 位 Linux 系统上的安装。

```
[root@localhost ~]# yum -y install openssl-devel
[root@localhost ~]# tar xzvf mongodb-linux-x86_64-rhel62-3.2.10.gz
[root@localhost ~]# mv mongodb-linux-x86_64-rhel62-3.2.10 /usr/local/mongodb
```

（2）创建 MongoDB 的数据存储目录 /data/mongodb1，日志存储目录 /data/logs，日志文件。

```
[root@localhost ~]# mkdir -p /data/mongodb1
[root@localhost ~]# mkdir -p /data/logs/mongodb
[root@localhost ~]# touch /data/logs/mongodb/mongodb1.log
[root@localhost ~]# chmod  -R 777 /data/logs/mongodb/mongodb1.log
```

当 MongoDB 处于频繁访问的状态，如果 shell 启动进程所占用的资源设置过低，将会产生错误导致无法连接 MongoDB 实例。需设置 ulimit-n 和 ulimit-u 的值大于 20000。

```
[root@localhost ~]# ulimit -n 25000
[root@localhost ~]# ulimit -u 25000
```

2. MongoDB 的运行控制

（1）创建 MongoDB 配置文件和启动参数

```
[root@localhost ~]# cd /usr/local/mongodb/bin/
[root@localhost bin]# vim mongodb1.conf
```

```
port=27017                                    //默认服务器端口号
dbpath=/data/mongodb1                         //数据存储目录
logpath=/data/logs/mongodb/mongodb1.log       //日志文件
logappend=true                                //使用追加方式写日志
fork=true                                     //后台运行
maxConns=5000                                 //最大同时连接数，默认 2000
storageEngine=mmapv1                          //指定存储引擎为内存映射文件
```

（2）设置内核参数

当某个节点可用内存不足时，系统会从其他节点分配内存。

```
[root@localhost bin]# echo 0 > /proc/sys/vm/zone_reclaim_mode
 [root@localhost bin]# sysctl -w vm.zone_reclaim_mode=0    //永久设置
[root@localhost~]# echo never > /sys/kernel/mm/transparent_hugepage/enabled
[root@localhost~]# echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

9.2.2 启动和停止 MongoDB

安装好后就可以启动 MongoDB 服务了。

1. 启动 MongoDB

启动 MongoDB 的命令如下：

```
[root@localhost~]#/usr/local/mongodb/bin/mongod -f \
/usr/local/mongodb/bin/mongodb1.conf
```

或者

```
[root@localhost~]#/usr/local/mongodb/bin/mongod --config \
/usr/local/mongodb/bin/mongodb1.conf
[root@localhost bin]# netstat -antp | grep mongod
tcp   0  0 0.0.0.0:27017     0.0.0.0:*         LISTEN      3153/mongod
```

2. 停止 MongoDB

停止 MongoDB 的命令如下：

```
[root@localhost~]# /usr/local/mongodb/bin/mongod --config \
/usr/local/mongodb/bin/mongodb1.conf --shutdown
```

或者

```
[root@localhost~]# ps aux | grep mongodb
root   1213  1.2  1.4 531076 54916 ?        Sl   01:40   0:00 /usr/local/mongodb/bin/mongod -f /usr/
    local/mongodb/bin/mongodb1.conf
[root@localhost~]# kill -2 1213
```

3. 配置开机自动启动

配置 MongoDB 自动启动的命令如下：

```
[root@localhost~]# vi /etc/rc.local
rm -rf /data/mongodb1/mongod.lock
/usr/local/mongodb/bin/mongod --config /usr/local/mongodb/bin/mongodb1.conf
```

配置完成后重启系统，MongoDB 会自动启动。

4. 连接并访问 MongoDB

启动 MongoDB 后就可以通过 MongoDB 客户端工具连接数据库了。

```
[root@localhost~]# /usr/local/mongodb/bin/mongo
> use admin
switched to db admin
> db.shutdownServer()                    // 也可以通过这种方式关闭数据库
server should be down...
```

9.2.3　启动 MongoDB 多实例

在单台服务器资源充分的情况下，可以使用多实例，以便充分使用服务器资源。具体步骤如下：

```
[root@localhost ~]# cd /usr/local/mongodb/bin/
[root@localhost bin]# cp mongodb1.conf mongodb2.conf
[root@localhost bin]# vim mongodb2.conf
port=27018
dbpath=/data/mongodb2
logpath=/data/logs/mongodb/mongodb2.log
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
 [root@localhost bin]# mkdir /data/mongodb2/
[root@localhost bin]# touch /data/logs/mongodb/mongodb2.log
[root@localhost bin]# chmod -R 777 /data/logs/mongodb/mongodb2.log
[root@localhost bin]# ./mongod -f mongodb2.conf
about to fork child process, waiting until server is ready for connections.
forked process: 1208
child process started successfully, parent exiting
[root@localhost bin]#
```

9.3　MongoDB 的存储结构

本节将介绍 MongoDB 的逻辑存储结构和物理存储结构，以及 MongoDB 的常见数据类型。

9.3.1 MongoDB 逻辑存储结构

MongoDB的逻辑结构主要由文档(document)、集合(collection)和数据库(database)三部分组成。其中文档是MongoDB的核心概念，它是MongoDB逻辑存储的最小单元，相当于关系型数据库中的一行记录，多个文档组成集合，集合相当于关系型数据库中的表的概念，多个集合组成数据库。

表9-1为MongoDB中的一些术语及说明。

表9-1 MongoDB的常用术语及说明

SQL 术语 / 概念	MongoDB 术语 / 概念	解释 / 说明
database	database	数据库
table	collection	数据库表 / 集合
row	document	数据记录行 / 文档
column	field	数据字段 / 域
index	index	索引
table joins		表连接，MongoDB 不支持
primary key	primary key	主键，MongoDB 自动将 _id 字段设置为主键

1. 数据库

一个MongoDB中可以建立多个数据库。MongoDB的默认数据库为test，该数据库存储在data目录中，要显示它，需要向数据库插入一些数据。MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。

```
[root@localhost ~]# alias mongo="/usr/local/mongodb/bin/mongo"
[root@Cent6S1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
> show dbs
local  0.078GB
> show databases;
local  0.078GB
> use local;
switched to db local
>
```

有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

- admin：从权限的角度来看，这是root数据库。如果将一个用户添加到这个数据库，这个用户将自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。

- local：这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合。
- config：当 Mongo 用于分片设置时，config 数据库在内部使用，用于保存分片的相关信息。

2. 集合

集合就是 MongoDB 文档组，类似于关系数据库管理系统（Relational Database Management System，RDBMS）中的表格。

集合存在于数据库中，集合没有固定的结构，这意味着在集合中可以插入不同格式和类型的数据，但通常情况下插入集合的数据都会有一定的关联性。

```
> use local;
switched to db local
> show collections;
startup_log
system.indexes
> show tables;
startup_log
system.indexes
>
```

当第一个文档插入时，集合就会被创建。

合法的集合名不能是空字符串 ""，不能含有 \0 字符（空字符），这个字符表示集合名的结尾，不能以“system.”开头，这是为系统集合保留的前缀。

3. 文档

文档是一个键值（key-value）对，即 BSON。MongoDB 的文档不需要设置相同的字段，并且相同的字段不需要相同的数据类型，这与关系型数据库有很大的区别，也是 MongoDB 非常突出的特点。

一个简单的文档例子如下：

"site":"www.kgc.com", "name":" 菜鸟教程 "

需要注意的是：

（1）文档中的键值对是有序的。

（2）文档中的值可以是双引号里面的字符串，也可以是其他几种数据类型（甚至可以是整个嵌入的文档）。

（3）MongoDB 区分类型和大小写。

（4）MongoDB 的文档不能有重复的键。

（5）文档的键是字符串。除了少数例外情况，键可以使用任意 UTF-8 字符。

文档键命名规范：

（1）键不能含有 \0（空字符），这个字符用来表示键的结尾。

（2）. 和 $ 有特别的意义，只有在特定环境下才能使用。

（3）以下划线“_”开头的键是保留的（不是严格要求的）。

下面示例演示了如何向集合中添加文档：

```
[root@Cent6S1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
> use kgc                                    // 创建并打开 kgc 数据库
switched to db kgc
> db.users.insert({"id":1,"name":"Tom"});   // 向 kgc 数据库的 users 集合添加一行数据
WriteResult({ "nInserted" : 1 })
> show collections                           // 显示 kgc 数据库中的所有集合
system.indexes
users
> db.users.insert({"id":2,"name":"Jerry","isadmin":true,"gender":null,
"favorite":["apple","banana","orange",1,2,3],"regtime":new Date()});
WriteResult({ "nInserted" : 1 })
> db.users.find()                    // 显示 users 集合中的所有文档
{ "_id" : ObjectId("583176f9fd7a2827d8ee2842"), "id" : 1, "name" : "Tom" }
{ "_id" : ObjectId("58317964fd7a2827d8ee2843"), "id" : 2, "name" : "Jerry",
 "isadmin" : true, "gender" : null, "favorite" : [ "apple", "banana", "orange",
1, 2, 3 ], "regtime" : ISODate("2016-11-20T10:22:28.432Z") }
>
```

9.3.2 MongoDB 物理存储结构

MongoDB 物理存储结构主要包括数据存储和日志存储。

1. 数据存储

MongoDB 的数据目录由配置文件中的 dbpath 指定，这里为 /data/mongodb1，用来存储所有 MongoDB 的数据文件。

```
[root@Cent6S1 bin]# ls -lh /data/mongodb1/
总用量 81M
drwxr-xr-x. 2 root root 4.0K 11 月 19 19:10 diagnostic.data
drwxr-xr-x. 2 root root 4.0K 11 月 18 02:17 journal
-rw-------. 1 root root 64M 11 月 18 02:17 kgc.0
-rw-------. 1 root root 16M 11 月 18 02:17 kgc.ns
-rw-------. 1 root root  64M 11 月 18 02:17 local.0
-rw-------. 1 root root  16M 11 月 18 02:17 local.ns
-rw-r--r--. 1 root root    5 11 月 18 02:17 mongod.lock
-rw-r--r--. 1 root root   69 11 月 17 22:01 storage.bson
[root@Cent6S1 bin]#
```

在 MongoDB 内部，每个数据库都包含一个 .ns 文件和一些数据文件，比如 local.ns、local.0，随着数据量不断增大，文件数量也会不断增多，每张表对应一个命令空间。

每个索引也有对应的命名空间。而这些命名空间的元数据都放在 .ns 文件中。

MongoDB 内部有预分配空间的机制，预分配的文件都用 0 进行填充，这样 MongoDB 始终保持额外的空间和空余的数据文件。数据文件每次新生成的一个文件，大小都会是上一个文件的两倍。

文件使用 MMAP 进行内存映射，会将所有数据文件映射到内存中，但是只是虚拟内存，只有访问到这块数据时才会交换到物理内存。

2. 日志存储

日志存储结构：

- 系统日志文件的存放由配置文件中的 logpath 指定，这里为 /data/logs/mongodb/。
- Journal 日志文件，用于 MongoDB 崩溃恢复的保障。
- oplog 复制操作日志文件在启动主从复制时出现。
- 慢查询日志文件，需要在配置文件中指定 profile=1（开启慢查询）与 slowms =200（记录毫秒数），查看慢查询的命令为：>db.system.profile.find()。

9.3.3　MongoDB 数据类型

表 9-2 为 MongoDB 中常用的几种数据类型。

表 9-2　MongoDB 中常用的数据类型

数据类型	描述
String	字符串。存储数据常用的数据类型。在 MongoDB 中 UTF-8 编码的字符串才是合法的
Integer	整型数值。用于存储数值，根据用户的服务器，可分为 32 位或 64 位
Boolean	布尔值。用于存储布尔值（真 / 假）
Double	双精度浮点值。用于存储浮点值
Arrays	用于将数组或列表或多个值存储为一个键
Object	用于内嵌文档
Null	用于创建空值
Date	日期时间。用户可以指定自己的日期时间，创建 Date 对象，传入年月日信息
Binary Data	二进制数据。用于存储二进制数据

下面示例演示了如何查看 MongoDB 的数据类型。

```
> a=db.users.findOne({“id”:2});
{
    "_id" : ObjectId("58317964fd7a2827d8ee2843"),
    "id" : 2,
    "name" : "Jerry",
    "isadmin" : true,
```

```
        "gender" : null,
        "favorite" : [
                "apple",
                "banana",
                "orange",
                1,
                2,
                3
        ],
        "regtime" : ISODate("2016-11-20T10:22:28.432Z")
}
> typeof(ia.id)
number
> typeof(a.name)
string
> typeof(a.isadmin)
boolean
> typeof(a.gender)
object
> typeof(a.favorite)
undefined
> typeof(a.favorate)
object
> typeof(a.regtime)
object
>
```

9.4 MongoDB 基本操作

MongoDB 的基本操作主要是对数据库、集合、文档的操作，包括创建数据库、删除数据库、插入文档、更改文档、删除文档和查询文档。

表 9-3 列出了对数据库的一些常用操作。

表 9-3 MongoDB 中常用的基本操作

操作	描述
show dbs	查看当前实例下的数据库列表
show users	显示用户
use <db_name>	切换当前数据库
db.help()	显示数据库操作命令
show collections	显示当前数据库中的集合
db.foo.help()	显示集合操作命令，foo 是当前数据库下的集合
db.foo.find()	对当前数据库中 foo 集合进行数据查找

9.4.1　在 MongoDB 中创建和删除数据库

MongoDB 创建数据库的语法格式如下：

```
use DATABASE_NAME
```

如果数据库不存在，则创建数据库，否则切换到指定数据库。

下面我们创建数据库 kgc1，然后通过 db.dropDatabase() 命令删除数据库，最后再通过 show dbs 命令查看数据库是否删除成功。

```
> use kgc1
switched to db kgc1
> db
kgc1
> show dbs
kgc1 0.078GB
local  0.078GB
> db.dropDatabase()
{ "dropped" : "kgc1", "ok" : 1 }
> show dbs
local  0.078GB
>
```

9.4.2　MongoDB 的集合和文档操作

数据保存在 MongoDB 的集合中。所有存储在集合中的数据都是 Binary JSON 格式，简称 BSON。BSON 是一种类似于 JSON 的二进制形式的存储格式。

MongoDB 使用 insert() 方法向集合中插入文档，语法如下：

```
db.COLLECTION_NAME.insert(document)
```

同样，MongoDB 使用 update() 方法来更新集合中的文档。使用 remove() 方法来移除集合中的数据。

```
> use kgc
switched to db kgc
> show collections
system.indexes
users
> db.users.find()
{ "_id" : ObjectId("58318ba2fd7a2827d8ee2844"), "id" : 1, "name" : "Tom" }
{ "_id" : ObjectId("58318bc7fd7a2827d8ee2845"), "id" : 2, "name" : "Jerry",
"isadmin" : true, "gender" : null, "favorate" : [ "apple", "banana", "orange",
1, 2, 3 ], "regtime" : ISODate("2016-11-20T11:40:55.445Z") }
> db.users.update({"id":1},{$set:{"name":"Bob"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.users.find()
{ "_id" : ObjectId("58318ba2fd7a2827d8ee2844"), "id" : 1, "name" : "Bob" }
{ "_id" : ObjectId("58318bc7fd7a2827d8ee2845"), "id" : 2, "name" : "Jerry",
"isadmin" : true, "gender" : null, "favorate" : [ "apple", "banana", "orange",
1, 2, 3 ], "regtime" : ISODate("2016-11-20T11:40:55.445Z") }
> db.users.remove({"id":2})
WriteResult({ "nRemoved" : 1 })
> db.users.find()
{ "_id" : ObjectId("58318ba2fd7a2827d8ee2844"), "id" : 1, "name" : "Bob" }
> show collections
system.indexes
users
> db.users.drop()
true
> show collections
system.indexes
>
```

9.5 MongoDB 日常维护

MongoDB 日常维护主要包括数据库的备份恢复、安全管理和数据库状态监控。

9.5.1 备份与恢复管理

在 MongoDB 中备份管理包括导入导出、备份与恢复、复制数据库和克隆集合等操作。

1. 导入导出

可以使用 mongoexport 和 mongoimport 命令来导出导入 MongoDB 的数据。

mongoexport 命令可以把一个 collection 导出成 JSON 格式或 CSV 格式的文件。可以通过参数指定导出的数据项，也可以根据指定的条件导出数据。

mongoimport 命令可以把一个特定格式文件中的内容导入到指定的 collection 中。该工具可以导入 JSON 格式的数据，也可以导入 CSV 格式的数据。

mongoexport 和 mongoimport 具体用法如下所示：

```
> for(var i=1;i<=10000;i++) db.users.insert({"id":i,"name":"JackyALice"});
WriteResult({ "nInserted" : 1 })
> db.users.count();
10000
> db
kgc
> exit
bye
```

```
[root@Cent6S1 bin]# ./mongoexport -d kgc -c users -o ./users.json
2016-11-20T21:51:13.603+0800  connected to: localhost
2016-11-20T21:51:14.078+0800  exported 10000 records
[root@Cent6S1 bin]# ll u*
-rw-r--r--. 1 root root 858894 11 月 20 21:51 users.json
[root@Cent6S1 bin]# head -n 3 users.json
{"_id":{"$oid":"5831a94cfd7a2827d8ee2846"},"id":1.0,"name":"JackyALice"}
{"_id":{"$oid":"5831a94cfd7a2827d8ee2847"},"id":2.0,"name":"JackyALice"}
{"_id":{"$oid":"5831a94cfd7a2827d8ee2848"},"id":3.0,"name":"JackyALice"}
[root@Cent6S1 bin]# ./mongoimport -d kgc -c user1 --file users.json
2016-11-20T21:53:15.853+0800  connected to: localhost
2016-11-20T21:53:16.147+0800  imported 10000 documents
[root@Cent6S1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
> use kgc
switched to db kgc
> show collections
system.indexes
user1
users
> exit
bye
[root@r6s1 bin]# ./mongoexport -d kgc -c user1 \
> -q '{"id":{"$gt":5000}}' -o user2.json;
2016-07-09T18:49:43.555+0800  connected to: localhost
2016-07-09T18:49:44.397+0800  exported 5000 records
[root@r6s1 bin]#
```

参数说明：

- -d：指明数据库的名字。
- -c：指明 collection 的名字。
- -f：指明要导出哪些列。
- -o：指明要导出的文件名。
- -q：指明导出数据的过滤条件。

具体命令帮助信息可以通过 --help 命令查看。

2. 备份与恢复

在 MongoDB 中可以使用 mongodump 命令来备份 MongoDB 数据，该命令可以导出所有数据到指定目录中。mongodump 命令可以通过参数指定导出的数据量级转存的服务器。

mongodump 命令脚本语法如下：

```
>mongodump -h dbhost -d dbname -o dbdirectory
```

参数说明：

- -h：MongoDB 所在服务器地址，例如：127.0.0.1，当然也可以指定端口号，如：

127.0.0.1:27017。

- -d：需要备份的数据库实例，例如：test。
- -o：备份的数据存放的位置，该目录需要提前建立，在备份完成后，系统自动在 dump 目录下建立一个 test 目录，这个目录里面存放该数据库实例的备份数据。

```
[root@Cent6S1 bin]# mkdir backup
[root@Cent6S1 bin]# ./mongodump -d kgc -o ./backup/
[root@Cent6S1 bin]# ls -lth ./backup/kgc/
总用量 1.3M
-rw-r--r--. 1 root root 635K 11 月 20 22:09 user1.bson
-rw-r--r--. 1 root root 635K 11 月 20 22:09 users.bson
-rw-r--r--. 1 root root  128 11 月 20 22:09 system.indexes.bson
-rw-r--r--. 1 root root   81 11 月 20 22:09 user1.metadata.json
-rw-r--r--. 1 root root   81 11 月 20 22:09 users.metadata.json
[root@Cent6S1 bin]#
```

MongoDB 使用 mongorestore 命令来恢复备份的数据。语法如下：

```
mongorestore -h dbhost -d dbname --directoryperdb dbdirectory
[root@Cent6S1 bin]# ./mongorestore -d kgc1 --dir=./backup/kgc/
```

3. 复制数据库

在 MongoDB 中可以使用 db.copyDatabase 命令复制数据库。

```
> show dbs
kgc   0.078GB
kgc1  0.078GB
local 0.078GB
> db.copyDatabase("kgc","kgc2","localhost");
{ "ok" : 1 }
> show dbs
kgc   0.078GB
kgc1  0.078GB
kgc2  0.078GB
local 0.078GB
>
```

4. 克隆集合

在 MongoDB 中可以将数据库中的集合进行克隆。这里将 kgc 数据库中的 user1 集合克隆到另外一个实例。

```
> use kgc
switched to db kgc
> show collections
system.indexes
user1
users
```

```
> exit
bye
[root@Cent6S1 bin]# ./mongod -f mongodb2.conf
about to fork child process, waiting until server is ready for connections.
forked process: 2262
child process started successfully, parent exiting
[root@Cent6S1 bin]# netstat -antp | grep mongod
tcp  0  0 0.0.0.0:27017    0.0.0.0:*           LISTEN      1180/mongod
tcp  0  0 0.0.0.0:27018    0.0.0.0:*           LISTEN      2262/./mongod
[root@Cent6S1 bin]# mongo --port 27018
MongoDB shell version: 3.2.10
connecting to: 127.0.0.1:27018/test
> show dbs
local  0.078GB
> db.runCommand({"cloneCollection":"kgc.user1","from":"192.168.6.110:27017"});
{ "ok" : 1 }
> show dbs
kgc    0.078GB
local  0.078GB
> use kgc
switched to db kgc
> show collections
system.indexes
user1
>
```

9.5.2　安全管理

MongoDB 安全管理主要包括 MongoDB 的安全访问控制和用户权限分配。

1. 限定监听特定 IP 和端口

只绑定内网卡地址：bind_ip=192.168.6.110。

只侦听指定的端口：port=27017。

```
[root@CentOS6-node1 bin]# vim mongodb1.conf
bind_ip=192.168.6.110
port=27017
......
[root@CentOS6-node1 bin]# netstat -antp | grep 27017
tcp    0  0 0.0.0.0:27017        0.0.0.0:*           LISTEN      3163/mongod
[root@CentOS6-node1 bin]# ./mongod -f mongodb1.conf -shutdown
[root@CentOS6-node1 bin]# ./mongod -f mongodb1.conf
[root@CentOS6-node1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
......
exception: connect failed
[root@CentOS6-node1 bin]# mongo --host 192.168.6.110
```

```
MongoDB shell version: 3.2.10
connecting to: 192.168.6.110:27017/test
> show dbs
local  0.078GB
```

2. 授权启动

可以配置授权用户来访问 MongoDB，启动时必须指定 auth=true，否则授权不起作用。

```
[root@CentOS6-node1 bin]# mongo --host 192.168.6.110
MongoDB shell version: 3.2.10
connecting to: 192.168.6.110:27017/test
> use admin
switched to db admin
> db.createUser({"user":"root",
... "pwd":"123",
... "roles":["root"]})
Successfully added user: { "user" : "root", "roles" : [ "root" ] }
> exit
bye
[root@CentOS6-node1 bin]# vim mongodb1.conf
bind_ip=192.168.6.110
port=27017
auth=true
......
[root@CentOS6-node1 bin]# mongo --host 192.168.6.110
MongoDB shell version: 3.2.10
connecting to: 192.168.6.110:27017/test
> show dbs
2016-07-11T15:09:23.216+0800 E QUERY   [thread1] Error: listDatabases failed:{
    "ok" : 0,
    "errmsg" : "not authorized on admin to execute command { listDatabases:
1.0 }",
    "code" : 13
} :
......
> use admin
switched to db admin
> db.auth("root","123")
1
> show dbs
admin  0.078GB
local  0.078GB
>
```

可以将用户加入到角色，内置数据库用户角色包括：read、readWrite，数据库管理角色包括：dbAdmin、dbOwner、userAdmin，超级用户角色为 root。

3. 进程管理

管理员可以对 MongoDB 进程进行管理和控制。

查看当前正在运行的进程的命令为：db.currentOp()。

终止正在运行的高消耗资源的进程的命令为：db.killOp(opid)。

```
> db.currentOp()
{
  "inprog" : [
    {
      "desc" : "conn1",
      "threadId" : "140433909839616",
      "connectionId" : 1,
      "client" : "192.168.6.110:53924",
      "active" : true,
      "opid" : 4872,
      "secs_running" : 0,
      "microsecs_running" : NumberLong(31),
      "op" : "command",
      "ns" : "admin.$cmd",
      "query" : {
        "currentOp" : 1
      },
      "numYields" : 0,
      "locks" : {
      },
      "waitingForLock" : false,
      "lockStats" : {
      }
    }
  ],
  "ok" : 1
}
> db.killOp(4872)
{ "info" : "attempting to kill op", "ok" : 1 }
>
```

9.5.3　MongoDB 监控

MongoDB 提供一些内置工具可以监控数据库的状态信息。

1. 查看数据库统计信息的命令

- 查看数据库实例的状态信息：db.serverStatus()。
- 查看当前数据库的统计信息：db.stats()。

此外，也可以通过 Web 界面查看系统监控信息，需要在配置文件中添加下面一行：

```
httpinterface=true
```

然后在浏览器中通过 http://localhost:28017 进行访问。

通过 Web 页面可以看到：

（1）当前 MongoDB 的所有连接。

（2）各个数据库和 Collection 的访问统计，包括：Reads、Writes、Queries、GetMores、Inserts、Updates、Removes。

（3）写锁的状态。

（4）日志文件的最后几百行。

（5）所有的 MongoDB 命令。

2. 查看集合统计信息的命令

- 查看集合统计信息：db.users.stats()。
- 查看集合大小：db.users.dataSize()。

3. 第三方监控工具

可以在 Nagios 中配置使用 MongoDB 插件来监控 MongoDB 数据库。

本章总结

- MongoDB 是一款跨平台、面向文档的数据库。可以实现高性能、高可用性，并且能够轻松扩展。
- MongoDB 提供了复制、高可用性和自动分片功能，支持丰富的查询表达式，查询指令使用 JSON 形式的标记。
- MongoDB 的逻辑结构主要由文档（document）、集合（collection）和数据库（database）三部分组成。其中文档是 MongoDB 的核心概念。
- MongoDB 的基本操作主要是对数据库、集合、文档的操作，包括创建数据库、删除数据库、插入文档、更改文档、删除文档和查询文档。
- MongoDB 提供一些内置工具可以监控数据库的状态信息，如 db.serverStatus、db.stats()。

本章作业

1. 在 CentOS 平台上搭建 MongoDB 数据库，实现对 MongoDB 数据库的连接、创建、访问和安全设置，并将数据迁移到另外一个 MongoDB 实例。

2. 用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



第10章

MongoDB 复制集

技能目标

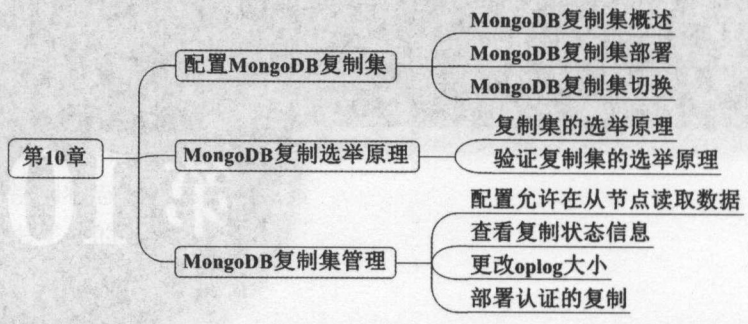
- 理解 MongoDB 复制集的概念
- 学会部署 MongoDB 复制集
- 理解 MongoDB 复制集的选举过程
- 学会 MongoDB 复制集的管理和维护

本章导读

MongoDB 复制集是将数据同步在多个服务器的过程。复制集提高了数据的可用性，并可以保证数据的安全性。

知识服务





10.1 配置 MongoDB 复制集

本节将依次介绍 MongoDB 复制集的基本概念、复制集的工作原理和复制集的部署。

10.1.1 MongoDB 复制集概述

1. 什么是复制集

复制集（Replica Sets）是额外的数据副本，是跨多个服务器同步数据的过程，复制集提供了冗余并增加了数据可用性，通过复制集可以对硬件故障和中断的服务进行恢复。

复制集的优势如下：

- 让数据更安全。
- 高数据可用性（24*7）。
- 灾难恢复。
- 无停机维护（如备份、索引重建、故障转移）。
- 读缩放（额外的副本读取）。
- 副本集对应用程序是透明的。

2. 复制集工作原理

MongoDB 的复制集至少需要两个节点。其中一个是主节点（Primary），负责处理客户端的请求，其余的都是从节点（Secondary），负责复制主节点上的数据。

Mongodb 各个节点常见的搭配方式为：一主一从或一主多从。主节点记录其上的所有操作到 oplog 中，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。

MongoDB 复制结构图如图 10.1 所示。

客户端在主节点写入数据，在从节点读取数据，主节点与从节点进行数据交互保障数据的一致性。如果其中一个节点出现故障，其他节点马上会将业务接过来而无需停机操作。

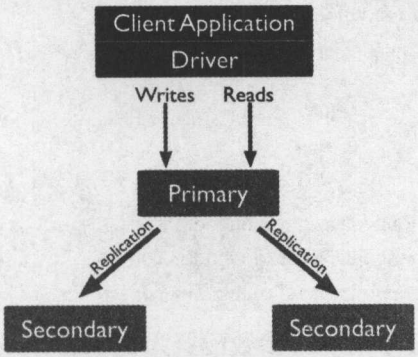


图 10.1　MongoDB 复制结构图

复制集特点如下：

- N 个节点的群集。
- 任何节点可作为主节点。
- 所有写入操作都在主节点上。
- 自动故障转移。
- 自动恢复。

10.1.2　MongoDB 复制集部署

下面将一步一步介绍如何部署 MongoDB 复制集，包括复制集的初始化配置、启动复制集和增加、删除复制集的节点。

1. 配置复制集

上一章已经搭建配置过 2 个 MongoDB 实例 mongodb1 和 mongodb2，在此基础上我们再添加并配置 mongodb3 和 mongodb4 这两个实例。

（1）创建数据文件和日志文件存储路径

```
[root@localhost ~]# mkdir -p /data/mongodb{3,4}/
[root@localhost ~]# touch /data/logs/mongodb/mongodb{3,4}.log
[root@localhost ~]# chmod -R 777 /data/logs/mongodb/*.log
```

（2）编辑 4 个 MongoDB 实例的配置文件

先编辑 Mongodb1 的配置文件，配置 replSet 参数值都为 kgcrs，并复制 3 份，具体操作如下：

```
[root@localhost ~]# cd /usr/local/mongodb/bin/
[root@localhost bin]# vim mongodb1.conf
#bind_ip=192.168.6.110
port=27017
#auth=true
dbpath=/data/mongodb1
```

```
logpath=/data/logs/mongodb/mongodb1.log
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
replSet=kgcrs
[root@localhost bin]# cp mongodb1.conf mongodb2.conf
[root@localhost bin]# cp mongodb1.conf mongodb3.conf
[root@localhost bin]# cp mongodb1.conf mongodb4.conf
```

然后将 mongodb2.conf 中的 port 参数配置为 27018，mongodb3.conf 中的 port 参数配置为 27019，mongodb4.conf 中的 port 参数配置为 27020。同样也将 dbpath 和 logpath 参数修改为对应的路径值。

（3）编写启动停止 MongoDB 脚本，启动 4 个 MongoDB 节点实例

为了方便以后启动停止 MongoDB 实例，可以编写一个脚本。

```
[root@localhost bin]# vi mongodb
#!/bin/bash
INSTANCE=$1
ACTION=$2
case "$ACTION" in
'start')
/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"$INSTANCE".conf;;
'stop')
/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"$INSTANCE".conf --shutdown;;
'restart')
/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"$INSTANCE".conf --shutdown
/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/"$INSTANCE".conf;;
esac
[root@localhost bin]# chmod +x mongodb
[root@localhost bin]# ./mongodb mongodb1 restart
[root@localhost bin]# ./mongodb mongodb2 restart
[root@localhost bin]# ./mongodb mongodb3 restart
[root@localhost bin]# ./mongodb mongodb4 restart
[root@localhost bin]# netstat -anpt |grep mongod
tcp   0   0 0.0.0.0:27019     0.0.0.0:*       LISTEN     3257/mongod
tcp   0   0 0.0.0.0:27020     0.0.0.0:*       LISTEN     3216/mongod
tcp   0   0 0.0.0.0:27017     0.0.0.0:*       LISTEN     3154/mongod
tcp   0   0 0.0.0.0:27018     0.0.0.0:*       LISTEN     3175/mongod
```

（4）初始化配置复制集

启动完 4 个 MongoDB 实例后，下面介绍如何配置并启动 MongoDB 复制集。这里先配置包含 3 个节点的复制集。

```
[root@localhost bin]# alias mongo="/usr/local/mongodb/bin/mongo"
[root@localhost bin]# mongo
```

```
> show dbs
2016-11-27T20:39:51.167+0800 E QUERY    [thread1] Error: listDatabases failed:{ "ok" : 0, "errmsg" :
    "not master and slaveOk=false", "code" : 13435 } :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:62:1
shellHelper.show@src/mongo/shell/utils.js:761:19
shellHelper@src/mongo/shell/utils.js:651:15
@(shellhelp2):1:1
> rs.status()
{
    "info" : "run rs.initiate(...) if not yet done for the set",
    "ok" : 0,
    "errmsg" : "no replset config has been received",
    "code" : 94
}
> cfg={"_id":"kgcrs","members":[{"_id":0,"host":"192.168.6.110:27017"},
... {"_id":1,"host":"192.168.6.110:27018"},
... {"_id":2,"host":"192.168.6.110:27019"}]}
{
    "_id" : "kgcrs",
    "member" : [
        {
            "_id" : 0,
            "host" : "192.168.6.110:27017"
        },
        {
            "_id" : 1,
            "host" : "192.168.6.110:27018"
        },
        {
            "_id" : 2,
            "host" : "192.168.6.110:27019"
        }
    ]
}
> rs.initiate(cfg)
{ "ok" : 1 }
kgcrs:OTHER>
kgcrs:PRIMARY>
>
```

这里先通过 rs.status() 命令查看复制集的状态信息，提示复制集还未配置，接着定义 cfg 初始化参数，最后通过 rs.initiate(cfg) 命令启动复制集。

（5）查看复制集状态

启动复制集后，再次通过 rs.status() 命令查看复制集的完整状态信息。

```
kgcrs:PRIMARY> rs.status()
{
  "set" : "kgcrs",
  "date" : ISODate("2016-11-27T13:05:19.974Z"),
  "myState" : 1,
  "term" : NumberLong(1),
  "heartbeatIntervalMillis" : NumberLong(2000),
  "members" : [
    {
      "_id" : 0,
      "name" : "192.168.6.110:27017",
      "health" : 1,
      "state" : 1,
      "stateStr" : "PRIMARY",
      "uptime" : 391,
      "optime" : {
        "ts" : Timestamp(1480251828, 2),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2016-11-27T13:03:48Z"),
      "infoMessage" : "could not find member to sync from",
      "electionTime" : Timestamp(1480251828, 1),
      "electionDate" : ISODate("2016-11-27T13:03:48Z"),
      "configVersion" : 1,
      "self" : true
    },
    {
      "_id" : 1,
      "name" : "192.168.6.110:27018",
      "health" : 1,
      "state" : 2,
      "stateStr" : "SECONDARY",
      "uptime" : 103,
      "optime" : {
        "ts" : Timestamp(1480251828, 2),
        "t" : NumberLong(1)
      },
      "optimeDate" : ISODate("2016-11-27T13:03:48Z"),
      "lastHeartbeat" : ISODate("2016-11-27T13:05:18.360Z"),
      "lastHeartbeatRecv" : ISODate("2016-11-27T13:05:19.630Z"),
      "pingMs" : NumberLong(0),
      "syncingTo" : "192.168.6.110:27019",
      "configVersion" : 1
    },
    {
      "_id" : 2,
```

```
        "name" : "192.168.6.110:27019",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 103,
        "optime" : {
          "ts" : Timestamp(1480251828, 2),
          "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2016-11-27T13:03:48Z"),
        "lastHeartbeat" : ISODate("2016-11-27T13:05:18.360Z"),
        "lastHeartbeatRecv" : ISODate("2016-11-27T13:05:19.567Z"),
        "pingMs" : NumberLong(0),
        "syncingTo" : "192.168.6.110:27017",
        "configVersion" : 1
      }
    ],
    "ok" : 1
}
kgcrs:PRIMARY>
```

其中，health 为 1 代表健康，0 代表宕机。state 为 1 代表主节点，为 2 代表从节点。

注意

在复制集初始化配置时要保证从节点上没有数据。

2. 增加和删除节点

配置启动复制集后，还可以通过 rs.add() 和 rs.remove() 命令方便地添加或删除节点。

```
kgcrs:PRIMARY> rs.add("192.168.6.110:27020")
{ "ok" : 1 }
kgcrs:PRIMARY> rs.status()
{
    "set" : "kgcrs",
    "date" : ISODate("2016-11-27T13:12:36.943Z"),
    "myState" : 1,
    "term" : NumberLong(1),
    "heartbeatIntervalMillis" : NumberLong(2000),
    "members" : [

        ... ...

        {
            "_id" : 3,
            "name" : "192.168.6.110:27020",
```

```
            "health" : 1,
            "state" : 5,
            "stateStr" : "STARTUP2",
            "uptime" : 6,
            "optime" : {
               "ts" : Timestamp(0, 0),
               "t" : NumberLong(-1)
            },
            "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
            "lastHeartbeat" : ISODate("2016-11-27T13:12:35.050Z"),
            "lastHeartbeatRecv" : ISODate("2016-11-27T13:12:36.720Z"),
            "pingMs" : NumberLong(2),
            "syncingTo" : "192.168.6.110:27019",
            "configVersion" : 2
         }
      ],
      "ok" : 1
   }

   kgcrs:PRIMARY> rs.remove("192.168.6.110:27020")
   { "ok" : 1 }
   kgcrs:PRIMARY> rs.status()
```

10.1.3 MongoDB 复制集切换

MongoDB 复制集可以实现群集的高可用，当其中主节点出现故障时会自动切换到其他节点。管理员也可以手动进行复制集的主从切换。

1. 模拟故障自动转移

通过 kill 命令可以停止复制集的当前主节点，然后查看主节点会自动切换到其他节点上。

```
kgcrs:PRIMARY> exit
bye
[root@Cent6S1 bin]# ps aux | grep mongodb1
root      3562  0.4  3.1 5714900 124148 ?      Sl   20:58   0:14
/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/bin/mongodb1.conf
root      4233  0.0  0.0 103248   872 pts/1    S+   21:49   0:00 grep mongodb1
[root@Cent6S1 bin]# kill -9 3562
[root@Cent6S1 bin]# ps aux | grep mongodb1
root      4235  0.0  0.0 103248   868 pts/1    S+   21:49   0:00 grep mongodb1
[root@Cent6S1 bin]# mongo --port 27018
kgcrs:PRIMARY> rs.status()
{
   "set" : "kgcrs",
```

```
"date" : ISODate("2016-11-27T13:51:42.812Z"),
"myState" : 1,
"term" : NumberLong(2),
"heartbeatIntervalMillis" : NumberLong(2000),
"members" : [
    {
        "_id" : 0,
        "name" : "192.168.6.110:27017",
        "health" : 0,
        "state" : 8,
        "stateStr" : "(not reachable/healthy)",
        "uptime" : 0,
        "optime" : {
            "ts" : Timestamp(0, 0),
            "t" : NumberLong(-1)
        },
        "optimeDate" : ISODate("1970-01-01T00:00:00Z"),
        "lastHeartbeat" : ISODate("2016-11-27T13:51:42.396Z"),
        "lastHeartbeatRecv" : ISODate("2016-11-27T13:49:51.880Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "Connection refused",
        "configVersion" : -1
    },
    {
        "_id" : 1,
        "name" : "192.168.6.110:27018",
        "health" : 1,
        "state" : 1,
        "stateStr" : "PRIMARY",
        "uptime" : 2906,
        "optime" : {
            "ts" : Timestamp(1480254602, 2),
            "t" : NumberLong(2)
        },
        "optimeDate" : ISODate("2016-11-27T13:50:02Z"),
        "infoMessage" : "could not find member to sync from",
        "electionTime" : Timestamp(1480254602, 1),
        "electionDate" : ISODate("2016-11-27T13:50:02Z"),
        "configVersion" : 3,
        "self" : true
    },
    {
        "_id" : 2,
        "name" : "192.168.6.110:27019",
        "health" : 1,
        "state" : 2,
```

```
            "stateStr" : "SECONDARY",
            "uptime" : 2885,
            "optime" : {
               "ts" : Timestamp(1480254602, 2),
               "t" : NumberLong(2)
            },
            "optimeDate" : ISODate("2016-11-27T13:50:02Z"),
            "lastHeartbeat" : ISODate("2016-11-27T13:51:42.394Z"),
            "lastHeartbeatRecv" : ISODate("2016-11-27T13:51:40.929Z"),
            "pingMs" : NumberLong(0),
            "syncingTo" : "192.168.6.110:27018",
            "configVersion" : 3
         }
      ],
      "ok" : 1
   }
   kgcrs:PRIMARY>
```

这里可以看到原“_id”为 0 的主节点出现故障后，主节点自动切换到“_id”为 1 的节点了。

2. 手动进行主从切换

管理员也可以手动进行复制集的主从切换，操作步骤如下：

```
kgcrs:PRIMARY> exit
bye
[root@Cent6S1 bin]# ./mongodb mongodb1 start
about to fork child process, waiting until server is ready for connections.
forked process: 4249
child process started successfully, parent exiting
[root@Cent6S1 bin]# mongo --port 27018
kgcrs:PRIMARY> rs.status()
kgcrs:PRIMARY> rs.freeze(30)
{ "ok" : 1 }
kgcrs:PRIMARY> rs.stepDown(60,30)
kgcrs:SECONDARY> rs.status()
kgcrs:SECONDARY>
```

其中，freeze(30) 为暂停 30 秒不参加选举；rs.stepDown(60,30) 告诉主节点交出主节点位置，然后维持从节点状态不少于 60 秒，同时等待 30 秒以使主节点和从节点日志同步。

再次查看复制集的状态信息，发现主节点已经切换到另外一个实例。

10.2 MongoDB 复制选举原理

本节将介绍 MongoDB 复制集选举的工作原理，并通过示例验证选举的过程。

10.2.1　复制集的选举原理

MongoDB 复制集的节点是通过选举产生主节点的，下面将介绍复制集节点间选举的过程。

1. 复制的原理

复制是基于操作日志 oplog，相当于 MySQL 中的二进制日志，只记录发生改变的记录。复制是将主节点的 oplog 日志同步并应用到其他从节点的过程。

2. 选举的原理

节点类型分为标准（host）节点、被动（passive）节点和仲裁（arbiter）节点。

（1）只有标准节点可能被选举为活跃（primary）节点，有选举权。被动节点有完整副本，不可能成为活跃节点，有选举权。仲裁节点不复制数据，不可能成为活跃节点，只有选举权。

（2）标准节点与被动节点的区别：priority 值高者是标准节点，低者则为被动节点。

（3）选举规则是票数高者获胜，priority 是优先权为 0 ～ 1000 的值，相当于额外增加 0 ～ 1000 的票数。选举结果：票数高者获胜；若票数相同，数据新者获胜。

MongoDB 复制集节点间选举如图 10.2 所示。

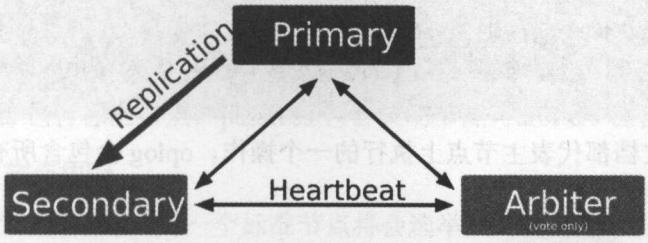


图 10.2　MongoDB 复制集节点间选举

10.2.2　验证复制集的选举原理

下面通过示例来演示 MongoDB 复制集节点间的选举原理。

1. 查看 oplog 日志

在 MongoDB 的 local 数据库中可以查看 oplog 日志，命令如下：

```
[root@Cent6S1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
kgcrs:PRIMARY> show dbs
kgc    0.078GB
kgc1   0.078GB
```

```
kgc2   0.078GB
local  2.077GB
kgcrs:PRIMARY> use kgc
kgcrs:PRIMARY> db.t1.insert({"id":1,"name":"Tom"})
kgcrs:PRIMARY> db.t1.insert({"id":2,"name":"Jerry"})
kgcrs:PRIMARY> db.t1.find()
kgcrs:PRIMARY> db.t1.update({"id":2},{"$set":{"name":"Bob"}})
kgcrs:PRIMARY> db.t1.remove({"id":2})
kgcrs:PRIMARY> use local
kgcrs:PRIMARY> show collections
kgcrs:PRIMARY> db.oplog.rs.find()
... ...
{ "ts" : Timestamp(1480258238, 1), "t" : NumberLong(4), "h" :
    NumberLong("1285436086348910797"), "v" : 2, "op" : "c", "ns" : "kgc.$cmd", "o" : { "create" : "t1" } }
{ "ts" : Timestamp(1480258238, 2), "t" : NumberLong(4), "h" : NumberLong("3058062851457454973"),
    "v" : 2, "op" : "i", "ns" : "kgc.t1", "o" : { "_id" : ObjectId("583af2be67db10c3c9d6ff8c"), "id" : 1,
    "name" : "Tom" } }
{ "ts" : Timestamp(1480258375, 1), "t" : NumberLong(4), "h" :
    NumberLong("-5851372469448433839"), "v" : 2, "op" : "i", "ns" : "kgc.t1", "o" : { "_id" : ObjectId("
    583af34767db10c3c9d6ff8d"), "id" : 2, "name" : "Jerry" } }
{ "ts" : Timestamp(1480258383, 1), "t" : NumberLong(4), "h" : NumberLong("5405713035253796 58"),
    "v" : 2, "op" : "u", "ns" : "kgc.t1", "o2" : { "_id" : ObjectId("583af34767db10c3c9d6ff8d") }, "o" : {
    "$set" : { "name" : "Bob" } } }
{ "ts" : Timestamp(1480258414, 1), "t" : NumberLong(4), "h" : NumberLong("8764883302575635102"),
    "v" : 2, "op" : "d", "ns" : "kgc.t1", "o" : { "_id" : ObjectId("583af34767db10c3c9d6ff8d") } }
kgcrs:PRIMARY>
```

其中的每个文档都代表主节点上执行的一个操作，oplog 会包含所有对数据有修改的操作（查询操作不会记录）。

2. 配置复制集的优先级

重新配置 4 个节点的 MongoDB 复制集，设置两个标准节点，一个被动节点和一个仲裁节点。命令如下：

```
kgcrs:PRIMARY> cfg={ “_id” :” kgcrs” ,” members” :[
... {"_id":0,"host":"192.168.6.110:27017","priority":100},
... {"_id":1,"host":"192.168.6.110:27018","priority":100},
... {"_id":2,"host":"192.168.6.110:27019","priority":0},
... {"_id":3,"host":"192.168.6.110:27020","arbiterOnly":true}]}
kgcrs:PRIMARY> rs.reconfig(cfg)
{ "ok" : 1 }
kgcrs:PRIMARY>
kgcrs:PRIMARY> rs.status()
kgcrs:PRIMARY> rs.isMaster()
{
   "hosts" : [
```

```
        "192.168.6.110:27017",
        "192.168.6.110:27018"
    ],
    "passives" : [
        "192.168.6.110:27019"
    ],
    "arbiters" : [
        "192.168.6.110:27020"
    ],
    "setName" : "kgcrs",
    "setVersion" : 5,
    "ismaster" : true,
    "secondary" : false,
    "primary" : "192.168.6.110:27017",
    "me" : "192.168.6.110:27017",
    "electionId" : ObjectId("7fffffff0000000000000004"),
    "maxBsonObjectSize" : 16777216,
    "maxMessageSizeBytes" : 48000000,
    "maxWriteBatchSize" : 1000,
    "localTime" : ISODate("2016-11-27T15:16:53.714Z"),
    "maxWireVersion" : 4,
    "minWireVersion" : 0,
    "ok" : 1
}
kgcrs:PRIMARY>
```

其中，hosts 包含标准节点，passives 包含被动节点，arbiters 包含仲裁节点。

3. 模拟主节点故障

如果主节点出现故障，另一个标准节点将会选举成为新的主节点。

```
kgcrs:PRIMARY> exit
bye
[root@Cent6S1 bin]# ./mongodb mongodb1 stop
killing process with pid: 4249
[root@Cent6S1 bin]# mongo --port 27018
MongoDB shell version: 3.2.10
connecting to: 127.0.0.1:27018/test
kgcrs:PRIMARY>
kgcrs:PRIMARY> rs.status()
......
        {
            "_id" : 1,
            "name" : "192.168.6.110:27018",
            "health" : 1,
            "state" : 1,
            "stateStr" : "PRIMARY",
```

```
        "uptime" : 10180,
        "optime" : {
          "ts" : Timestamp(1480261885, 1),
          "t" : NumberLong(5)
        },
        "optimeDate" : ISODate("2016-11-27T15:51:25Z"),
        "infoMessage" : "could not find member to sync from",
        "electionTime" : Timestamp(1480261884, 1),
        "electionDate" : ISODate("2016-11-27T15:51:24Z"),
        "configVersion" : 5,
        "self" : true
      },
.....
kgcrs:PRIMARY>
```

这里，当主节点为第一个标准节点出现故障后，MongoDB 复制集会选举第二个标准节点为主节点。

4. 模拟所有标准节点都出现故障

如果所有标准节点都出现故障，被动节点也不能成为主节点。

```
kgcrs:PRIMARY> exit
[root@Cent6S1 bin]# ./mongodb mongodb2 stop
killing process with pid: 3721
[root@Cent6S1 bin]# mongo --port 27019
MongoDB shell version: 3.2.10
connecting to: 127.0.0.1:27019/test
kgcrs:SECONDARY> rs.status()
.....
      {
        "_id" : 2,
        "name" : "192.168.6.110:27019",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 11089,
        "optime" : {
          "ts" : Timestamp(1480261885, 1),
          "t" : NumberLong(5)
        },
        "optimeDate" : ISODate("2016-11-27T15:51:25Z"),
        "infoMessage" : "could not find member to sync from",
        "configVersion" : 5,
        "self" : true
      },
.....
kgcrs:SECONDARY>
```

10.3　MongoDB 复制集管理

本节将介绍 MongoDB 复制集的基本配置和管理，分别包括配置从节点可以读取数据、查看复制集状态信息、更改 oplog 大小、配置带认证的复制集。

10.3.1　配置允许在从节点读取数据

默认 MongoDB 复制集的从节点不能读取数据，可以使用 rs.slaveOk() 命令允许能够在从节点读取数据。

重新启动两个标准节点后，连接到复制集的其中一个从节点，配置其允许读取数据。

```
[root@Cent6S1 bin]# ./mongodb mongodb1 start
[root@Cent6S1 bin]# ./mongodb mongodb2 start
[root@Cent6S1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
kgcrs:PRIMARY> rs.status()
kgcrs:PRIMARY> rs.help()
kgcrs:PRIMARY> use kgc
switched to db kgc
kgcrs:PRIMARY> show collections
system.indexes
t1
user1
users
kgcrs:PRIMARY> exit
bye
[root@Cent6S1 bin]# mongo --port 27018
MongoDB shell version: 3.2.10
connecting to: 127.0.0.1:27018/test
kgcrs:SECONDARY> show dbs
2016-11-28T01:17:44.874+0800 E QUERY    [thread1] Error: listDatabases failed:{ "ok" : 0, "errmsg" :
    "not master and slaveOk=false", "code" : 13435 } :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:62:1
shellHelper.show@src/mongo/shell/utils.js:761:19
shellHelper@src/mongo/shell/utils.js:651:15
@(shellhelp2):1:1

kgcrs:SECONDARY> rs.slaveOk()
kgcrs:SECONDARY> show dbs
kgc    0.078GB
```

```
kgc1   0.078GB
kgc2   0.078GB
local  2.077GB
kgcrs:SECONDARY>
```

10.3.2 查看复制状态信息

可以使用 rs.printReplicationInfo() 和 rs.printSlaveReplicationInfo() 命令来查看复制集状态。

```
kgcrs:SECONDARY> rs.help()
kgcrs:SECONDARY> rs.printSlaveReplicationInfo()
source: 192.168.6.110:27018
    syncedTo: Mon Nov 28 2016 01:07:28 GMT+0800 (CST)
    0 secs (0 hrs) behind the primary
source: 192.168.6.110:27019
    syncedTo: Mon Nov 28 2016 01:07:28 GMT+0800 (CST)
    0 secs (0 hrs) behind the primary
kgcrs:SECONDARY> rs.printReplicationInfo()
configured oplog size:   1574.91796875MB
log length start to end: 14632secs (4.06hrs)
oplog first event time:  Sun Nov 27 2016 21:03:36 GMT+0800 (CST)
oplog last event time:   Mon Nov 28 2016 01:07:28 GMT+0800 (CST)
now:                     Mon Nov 28 2016 01:21:17 GMT+0800 (CST)
kgcrs:SECONDARY>
kgcrs:SECONDARY> use local
switched to db local
kgcrs:SECONDARY> show collections
kgcrs:SECONDARY> db.system.replset.findOne()
```

10.3.3 更改 oplog 大小

oplog 即 operations log 的简写，存储在 local 数据库中。oplog 中新操作会自动替换旧的操作，以保证 oplog 不会超过预设的大小。默认情况下，oplog 大小会占用 64 位的实例 5% 的可用磁盘空间。

在 MongoDB 复制的过程中，主节点应用业务操作修改到数据库中，然后记录这些操作到 oplog 中，从节点复制这些 oplog，然后应用这些修改。这些操作是异步的。如果从节点的操作已经被主节点落下很远，oplog 日志在从节点还没执行完，oplog 可能已经轮滚一圈了，从节点跟不上同步，复制就会停下，从节点需要重新做完整的同步，为了避免此种情况，尽量保证主节点的 oplog 足够大，能够存放相当长时间的操作记录。

可以调用 db.runCommand 命令来更改 oplog 的大小。

```
kgcrs:SECONDARY> exit
```

```
bye
[root@Cent6S1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
kgcrs:PRIMARY> use local
kgcrs:PRIMARY> db.oplog.rs.stats()
{
   "ns" : "local.oplog.rs",
   "count" : 17,
   "size" : 1772,
   "avgObjSize" : 104,
   "numExtents" : 1,
   "storageSize" : 1758744576,
  ......
}
kgcrs:PRIMARY> rs.printReplicationInfo()
configured oplog size:   1677.26953125MB
log length start to end: 14632secs (4.06hrs)
oplog first event time:  Sun Nov 27 2016 21:03:36 GMT+0800 (CST)
oplog last event time:   Mon Nov 28 2016 01:07:28 GMT+0800 (CST)
now:                     Tue Nov 29 2016 02:24:06 GMT+0800 (CST)
kgcrs:PRIMARY> db.runCommand({"convertToCapped":"oplog.rs","size":102400000})
{ "ok" : 1 }
kgcrs:PRIMARY> db.oplog.rs.stats()
{
   "ns" : "local.oplog.rs",
   "count" : 17,
   "size" : 1772,
   "avgObjSize" : 104,
   "numExtents" : 1,
   "storageSize" : 102400000,
   ......
}
kgcrs:PRIMARY> rs.printReplicationInfo()
configured oplog size:   97.65625MB
log length start to end: 14632secs (4.06hrs)
oplog first event time:  Sun Nov 27 2016 21:03:36 GMT+0800 (CST)
oplog last event time:   Mon Nov 28 2016 01:07:28 GMT+0800 (CST)
now:                     Tue Nov 29 2016 02:26:57 GMT+0800 (CST)
kgcrs:PRIMARY>
```

10.3.4　部署认证的复制

下面示例演示了如何部署带用户身份认证的 MongoDB 复制集。

```
kgcrs:PRIMARY> use local
```

```
kgcrs:PRIMARY> db.createUser({"user":"root","pwd":"123","roles":["root"]})
Successfully added user: { "user" : "root", "roles" : [ "root" ] }
kgcrs:PRIMARY> exit
bye
[root@r6s1 bin]# vi mongodb1.conf          // 分别编辑 4 个配置文件，在后面添加 2 行
... ...
replSet=kgcrs
clusterAuthMode=keyFile
keyFile=/usr/local/mongodb/bin/kgcrskey1
[root@r6s1 bin]# vi mongodb2.conf
[root@r6s1 bin]# vi mongodb3.conf
[root@r6s1 bin]# vi mongodb4.conf
[root@r6s1 bin]# echo "kgcrs key" > kgcrskey1                    // 生成 4 个实例的密钥文件
[root@r6s1 bin]# echo "kgcrs key" > kgcrskey2
[root@r6s1 bin]# echo "kgcrs key" > kgcrskey3
[root@r6s1 bin]# echo "kgcrs key" > kgcrskey4
[root@r6s1 bin]# chmod 600 kgcrskey{1..4}
[root@r6s1 bin]# ./mongodb.service mongodb1 restart          // 重启 4 个实例
[root@r6s1 bin]# ./mongodb.service mongodb2 restart
[root@r6s1 bin]# ./mongodb.service mongodb3 restart
[root@r6s1 bin]# ./mongodb.service mongodb4 restart
[root@r6s1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: 192.168.6.110:27017/test
kgcrs:PRIMARY> show dbs
2016-11-28T14:04:28.688+0800 E QUERY    [thread1] Error: listDatabases failed:{
    "ok" : 0,
    "errmsg" : "not authorized on admin to execute command { listDatabases: 1.0 }",
    "code" : 13
} :
_getErrorWithCode@src/mongo/shell/utils.js:25:13
Mongo.prototype.getDBs@src/mongo/shell/mongo.js:62:1
shellHelper.show@src/mongo/shell/utils.js:760:19
shellHelper@src/mongo/shell/utils.js:650:15
@(shellhelp2):1:1

kgcrs:PRIMARY> rs.status()
{
    "ok" : 0,
    "errmsg" : "not authorized on admin to execute command { replSetGetStatus: 1.0 }",
    "code" : 13
}
kgcrs:PRIMARY> use admin
switched to db admin
kgcrs:PRIMARY> db.auth("root","123")
1
```

```
kgcrs:PRIMARY> rs.status()
kgcrs:PRIMARY> show dbs
kgc    0.078GB
kgc1   0.078GB
kgc2   0.078GB
local  2.077GB
kgcrs:PRIMARY >
```

本章总结

- MongoDB 复制集（Replica Sets）是额外的数据副本，复制集提供了冗余和增加数据可用性。
- MongoDB 的复制集至少需要两个节点。其中主节点负责处理客户端请求，从节点负责复制主节点上的数据。
- MongoDB 复制集可以实现群集的高可用，当主节点出现故障时会自动切换到其他节点。管理员也可以手动进行复制集的主从切换。
- 复制是基于操作日志 oplog，相当于 MySQL 中的二进制日志，只记录发生改变的记录。复制是将主节点的 oplog 日志同步并应用到其他从节点的过程。
- 节点类型分为标准节点、被动节点和仲裁节点。只有标准节点可能被选举为活跃（主）节点。
- 尽量保证主节点的 oplog 足够大，能够存放相当长时间的操作记录。

本章作业

1. 配置 MongoDB 复制集，添加或删除复制集的节点，模拟故障进行主从切换。
2. 配置复制集节点优先级，验证节点间的选举过程。
3. 配置允许从节点可以读取数据，配置带认证的复制集。
4. 用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



随手笔记



第11章

MongoDB 分片

技能目标

- 理解 MongoDB 分片的概念
- 学会部署 MongoDB 分片
- 理解 MongoDB 片键的选择
- 学会 MongoDB 分片管理和维护

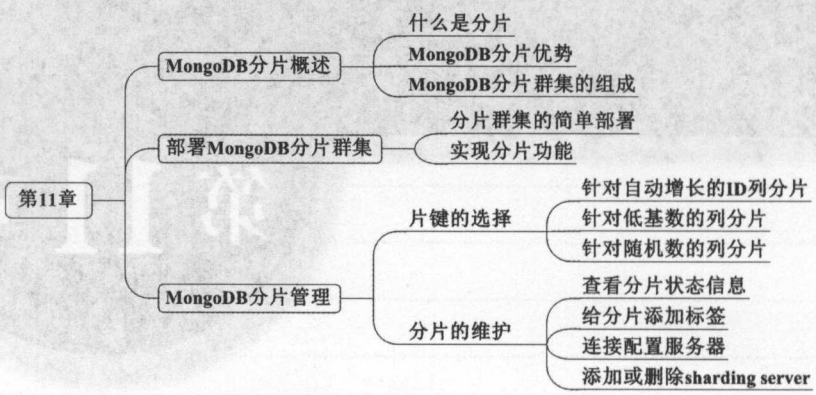
本章导读

分片是一种将海量数据水平扩展的数据库群集系统，数据分布存储在分片（sharding）的各个节点上，管理者通过简单的配置就可以很方便地构建一个分布式 MongoDB 分片群集。

复制集往往会与分片结合使用来保证 MongoDB 的高可用性。

知识服务





11.1 MongoDB 分片概述

本节将依次介绍 MongoDB 分片的基本概念、特点及分片群集的组成。

11.1.1 什么是分片

高数据量和吞吐量的数据库应用会对单机的性能造成较大压力，大的查询量会将单机的 CPU 耗尽，大的数据量对单机的存储压力较大，最终会耗尽系统的内存而将压力转移到磁盘 IO 上。

MongoDB 分片是使用多个服务器存储数据的方法，以支持巨大的数据存储和对数据进行操作。分片技术可以满足 MongoDB 数据量大量增长的需求，当一台 MongoDB 服务器不足以存储海量数据或不足以提供可接受的读写吞吐量时，我们就可以通过在多台服务器上分割数据，使得数据库系统能存储和处理更多的数据。

11.1.2 MongoDB 分片优势

分片为应对高吞吐量与大数据量提供了方法。

- 使用分片减少了每个分片需要处理的请求数，因此，通过水平扩展，群集可以提高自己的存储容量和吞吐量。比如，当插入一条数据时，应用只需要访问存储这条数据的分片。
- 使用分片减少了每个分片存储的数据。

MongoDB 分片结构如图 11.1 所示。

分片的优势在于提供类似线性增长的架构，提高数据可用性，提高大型数据库查询服务器的性能。当 MongoDB 单点数据库服务器存储成为瓶颈、单点数据库服务器的性能成为瓶颈或者需要部署大型应用以充分利用内存时，可以使用分片技术。

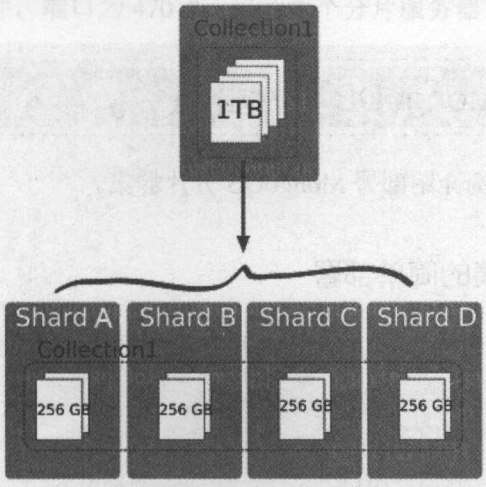


图 11.1　MongoDB 分片结构图

11.1.3　MongoDB 分片群集的组成

MongoDB 分片群集主要有如下三个主要组件：

- Shard：分片服务器，用于存储实际的数据块，实际生产环境中一个 shard server 角色可由几台服务器组成一个 Replica Set 承担，防止主机单点故障。
- Config Server：配置服务器，存储了整个分片群集的配置信息，其中包括 chunk 信息。
- Routers：前端路由，客户端由此接入，且让整个群集看上去像单一数据库，前端应用可以透明使用。

MongoDB 分片群集的组成如图 11.2 所示。

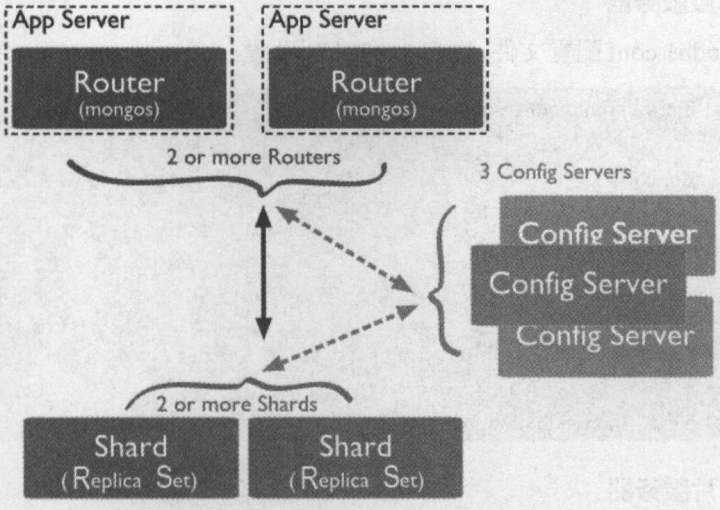


图 11.2　MongoDB 分片群集的组成

11.2 部署 MongoDB 分片群集

本节以一个简单案例介绍部署 MongoDB 分片群集。

11.2.1 分片群集的简单部署

下面介绍如何在一台物理服务器上部署一个简单结构的 MongoDB 分片群集。具体配置如下：

- 1 台路由实例（端口 27017）。
- 1 台配置实例（端口 37017）。
- 2 台 Shard 实例（端口 47017，47018）。

部署简单结构的 MongoDB 分片群集如图 11.3 所示。

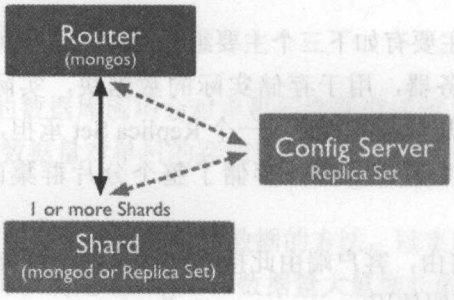


图 11.3 简单结构的 MongoDB 分片群集

1. 部署配置服务器

编辑 mongodb1.conf 配置文件，端口为 37017，设置 configsvr=true，启动配置服务器。

```
[root@cent6s1 bin]# vi mongodb1.conf
port=37017
dbpath=/data/mongodb1
logpath=/data/logs/mongodb/mongodb1.log
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
configsvr=true
[root@cent6s1 bin]# ./mongodb mongodb1 restart
```

2. 部署分片服务器

编辑 mongodb2.conf 配置文件，端口为 47017，设置 shardsvr=true。同样编辑

mongodb3.conf 配置文件，端口为 47018，启动两个分片服务器。

```
[root@cent6s1 bin]# vi mongodb2.conf
port=47017
dbpath=/data/mongodb2
logpath=/data/logs/mongodb/mongodb2.log
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
shardsvr=true
[root@cent6s1 bin]# vi mongodb3.conf
port=47018
dbpath=/data/mongodb3
logpath=/data/logs/mongodb/mongodb3.log
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
shardsvr=true
[root@cent6s1 bin]# ./mongodb mongodb2 restart
[root@cent6s1 bin]# ./mongodb mongodb3 restart
```

3. 启动路由服务器

通过 ./mongos --help 命令可以查看启动路由相关参数信息。chunkSize 为数据块大小，默认为 200MB，为了便于测试这里将值设置为 1。

```
[root@cent6s1 bin]# ./mongos --help
Options:
......
Sharding options:
  --configdb arg            Connection string for communicating
                            with config servers. Acceptable forms:
                            CSRS: <config replset
                            name>/<host1:port>,<host2:port>,[...]
                            SCCC (deprecated): <host1:port>,<host2:
                            port>,<host3:port>
                            Single-node (for testing only):
                            <host1:port>
  --localThreshold arg      ping time (in ms) for a node to be
                            considered local (default 15ms)
  --test                    just run unit tests
  --chunkSize arg           maximum amount of data per chunk
  --jsonp                   allow JSONP access via http (has
                            security implications)
  --noscripting             disable scripting engine
......
```

```
[root@cent6s1 bin]# ./mongos --port 27017 --fork \
> --logpath=/usr/local/mongodb/bin/route.log \
> --configdb 192.168.6.110:37017 --chunkSize 1
2016-11-25T19:42:31.465+0800 W SHARDING [main] Running a sharded cluster with fewer than 3
      config servers should only be done for testing purposes and is not recommended for production.
about to fork child process, waiting until server is ready for connections.
forked process: 30370
child process started successfully, parent exiting
[root@cent6s1 bin]#
```

4. 启用分片服务器

连接到路由实例后，可以通过 sh.status() 命令查看分片状态信息。

```
[root@cent6s1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
mongos> show dbs
config  0.031GB
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
}
  shards:
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
        No recent migrations
  databases:
mongos>
```

可以看到 shards 下目前没有分片服务器，通过 sh.addShard("192.168.6.110:47017") 命令添加分片服务器。

```
mongos> sh.addShard("192.168.6.110:47017")
{ "shardAdded" : "shard0000", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
```

```
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
}
  shards:
    { "_id" : "shard0000", "host" : "192.168.6.110:47017" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
        No recent migrations
  databases:
mongos>
```

添加第 2 个分片服务器，并查看分片信息。

```
mongos> sh.addShard( “192.168.6.110:47018” )
{ "shardAdded" : "shard0001", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
}
  shards:
    { "_id" : "shard0000", "host" : "192.168.6.110:47017" }
    { "_id" : "shard0001", "host" : "192.168.6.110:47018" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
        No recent migrations
  databases:
mongos>
```

11.2.2　实现分片功能

添加了两个分片服务器后，数据库和集合还未启用分片，这里先使用 mongoimport

命令导入 sales.txt 数据到 kgc 数据库的 sales 表。

```
mongos> exit
bye
[root@cent6s1 bin]# ./mongoimport -d kgc -c sales -f sid,qty,pid,price \
> --file sales.txt --type csv
2016-11-25T20:36:51.763+0800    connected to: localhost
2016-11-25T20:36:53.370+0800    imported 121317 documents
[root@cent6s1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
mongos> show dbs
config  0.031GB
kgc     0.078GB
mongos> use kgc
switched to db kgc
mongos> show collections
sales
system.indexes
mongos> db.sales.find().limit(5)
{ "_id" : ObjectId("58383063b89ebe8bbfe59206"), "sid" : 1, "qty" : 1, "pid" : 776, "price" : 2024.994 }
{ "_id" : ObjectId("58383063b89ebe8bbfe59207"), "sid" : 2, "qty" : 3, "pid" : 777, "price" : 2024.994 }
{ "_id" : ObjectId("58383063b89ebe8bbfe59208"), "sid" : 3, "qty" : 1, "pid" : 778, "price" : 2024.994 }
{ "_id" : ObjectId("58383063b89ebe8bbfe59209"), "sid" : 4, "qty" : 1, "pid" : 771, "price" : 2039.994 }
{ "_id" : ObjectId("58383063b89ebe8bbfe5920a"), "sid" : 5, "qty" : 1, "pid" : 772, "price" : 2039.994 }
mongos> db.sales.count()
121317
mongos>
```

使用 sh.status() 命令查看分片状态信息。"partitioned" 值为 false 表示数据库尚未分片。

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
}
  shards:
    {  "_id" : "shard0000",  "host" : "192.168.6.110:47017" }
    {  "_id" : "shard0001",  "host" : "192.168.6.110:47018" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
```

```
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
      No recent migrations
  databases:
    { "_id" : "kgc",  "primary" : "shard0000",  "partitioned" : false }
```

使用 sh.enableSharding("kgc") 命令启用 kgc 数据库分片。再次查看发现 "partitioned" 值为 true。

```
mongos> sh.enableSharding("kgc")
{ "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
  }
  shards:
    { "_id" : "shard0000",  "host" : "192.168.6.110:47017" }
    { "_id" : "shard0001",  "host" : "192.168.6.110:47018" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
      No recent migrations
  databases:
    { "_id" : "kgc",  "primary" : "shard0000",  "partitioned" : true }
mongos>
```

针对 sales 表创建索引，然后使用 sh.shardCollection("kgc.sales",{"pid":1}) 命令对 sales 集合进行分片。

```
mongos> db.sales.createIndex({"pid":1})
{
  "raw" : {
    "192.168.6.110:47017" : {
      "createdCollectionAutomatically" : false,
      "numIndexesBefore" : 1,
      "numIndexesAfter" : 2,
      "ok" : 1
    }
  },
```

```
    "ok" : 1
}
mongos> sh.shardCollection("kgc.sales",{"pid":1})
{ "collectionsharded" : "kgc.sales", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
}
  shards:
    {  "_id" : "shard0000",  "host" : "192.168.6.110:47017" }
    {  "_id" : "shard0001",  "host" : "192.168.6.110:47018" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
        13 : Success
  databases:
    {  "_id" : "kgc",  "primary" : "shard0000",  "partitioned" : true }
        kgc.sales
            shard key: { "pid" : 1 }
            unique: false
            balancing: true
            chunks:
                shard0000    13
                shard0001    13
            too many chunks to print, use verbose if you want to force print
mongos>
```

此时可以看到 chunks 均匀分布到两个分片上。

11.3 MongoDB 分片管理

本节将介绍 MongoDB 分片的管理，分别包括片键的选择、查看片键状态信息、添加多个分片、查看配置服务器信息以及添加删除分片服务器。

11.3.1 片键的选择

片键的选择原则上要结合实际的业务需求，尽可能选择存储分片均匀的键。

1. 针对自动增长的 ID 列分片

数据大多顺序地插入到一个分片上，这样可以提供无限数量的分片，但可能会造成单一且不可分散的热点。下面再次导入 sales 数据到 kgc 数据库的 sales2 集合。

```
mongos> exit
bye
[root@cent6s1 bin]# ./mongoimport -d kgc -c sales2 -f sid,qty,pid,price --file sales.txt --type csv
2016-11-25T22:12:46.905+0800    connected to: localhost
2016-11-25T22:12:48.742+0800    imported 121317 documents
[root@cent6s1 bin]# ./mongoimport -d kgc -c sales3 -f sid,qty,pid,price --file sales.txt --type csv
2016-11-25T22:13:00.024+0800    connected to: localhost
2016-11-25T22:13:01.970+0800    imported 121317 documents
```

2. 针对低基数的列分片

如 sales2 集合中的 qty 列进行分片。

```
[root@cent6s1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
mongos> use kgc
switched to db kgc
mongos> db.sales2.find().skip(5).limit(5)
{ "_id" : ObjectId("583846deb89ebe8bbfe76e4b"), "sid" : 6, "qty" : 2, "pid" : 773, "price" : 2039.994 }
{ "_id" : ObjectId("583846deb89ebe8bbfe76e4c"), "sid" : 7, "qty" : 1, "pid" : 774, "price" : 2039.994 }
{ "_id" : ObjectId("583846deb89ebe8bbfe76e4d"), "sid" : 8, "qty" : 3, "pid" : 714, "price" : 28.8404 }
{ "_id" : ObjectId("583846deb89ebe8bbfe76e4e"), "sid" : 9, "qty" : 1, "pid" : 716, "price" : 28.8404 }
{ "_id" : ObjectId("583846deb89ebe8bbfe76e4f"), "sid" : 10, "qty" : 6, "pid" : 709, "price" : 5.7 }
mongos> db.sales2.createIndex({"qty":1})
{
  "raw" : {
    "192.168.6.110:47017" : {
      "createdCollectionAutomatically" : false,
      "numIndexesBefore" : 1,
      "numIndexesAfter" : 2,
      "ok" : 1
    }
  },
  "ok" : 1
}
mongos> sh.shardCollection("kgc.sales2",{"qty":1})
{ "collectionsharded" : "kgc.sales2", "ok" : 1 }
```

查看分片状态信息。

```
mongos> sh.status()
--- Sharding Status ---
 sharding version: {
```

```
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
}
  shards:
    { "_id" : "shard0000", "host" : "192.168.6.110:47017" }
    { "_id" : "shard0001", "host" : "192.168.6.110:47018" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
        15 : Success
        3 : Failed with error 'chunk too big to move', from shard0000 to shard0001
  databases:
    { "_id" : "kgc", "primary" : "shard0000", "partitioned" : true }
        kgc.sales
            shard key: { "pid" : 1 }
            unique: false
            balancing: true
            chunks:
                shard0000   13
                shard0001   13
            too many chunks to print, use verbose if you want to force print
        kgc.sales2
            shard key: { "qty" : 1 }
            unique: false
            balancing: true
            chunks:
                shard0000   6
                shard0001   2
            { "qty" : { "$minKey" : 1 } } -->> { "qty" : 2 } on : shard0000 Timestamp(3, 1)
            { "qty" : 2 } -->> { "qty" : 3 } on : shard0000 Timestamp(1, 1) jumbo
            { "qty" : 3 } -->> { "qty" : 4 } on : shard0001 Timestamp(2, 0)
            { "qty" : 4 } -->> { "qty" : 5 } on : shard0001 Timestamp(3, 0)
            { "qty" : 5 } -->> { "qty" : 6 } on : shard0000 Timestamp(1, 4)
            { "qty" : 6 } -->> { "qty" : 7 } on : shard0000 Timestamp(1, 5)
            { "qty" : 7 } -->> { "qty" : 16 } on : shard0000 Timestamp(1, 6)
            { "qty" : 16 } -->> { "qty" : { "$maxKey" : 1 } } on : shard0000 Timestamp(1, 7)
mongos>
```

由于 qty 字段值有限，分片有限。

3. 针对随机数的列分片

下面再次导入 sales 数据到 kgc 数据库的 sales3 集合。针对 price 字段进行分片。

```
mongos> db.sales3.createIndex({ “price” :1})
mongos> sh.shardCollection("kgc.sales3",{"price":1})
mongos> sh.status()
......
      kgc.sales3
         shard key: { "price" : 1 }
         unique: false
         balancing: true
         chunks:
            shard0000    13
            shard0001    13
         too many chunks to print, use verbose if you want to force print
mongos>
```

price 字段值随机，可以有效分散热点，但数据越大，性能越差。
所以，片键的选择要结合业务的特点，可以使用组合的片键提高效率。

11.3.2　分片的维护

下面介绍如何对分片进行日常的维护。

1. 查看分片状态信息

使用 db.sales.stats() 命令可以查看 sales 集合的分片信息。

```
mongos> db.sales.stats()
{
   "sharded" : true,
   "paddingFactorNote" : "paddingFactor is unused and unmaintained in 3.0. It remains hard coded to
         1.0 for compatibility only.",
   "userFlags" : 1,
   "capped" : false,
   "ns" : "kgc.sales",
   "count" : 121317,
   "numExtents" : 13,
   "size" : 13587504,
   "storageSize" : 33689600,
   "totalIndexSize" : 9394224,
   "indexSizes" : {
      "_id_" : 5289872,
      "pid_1" : 4104352
   },
   "avgObjSize" : 112,
   "nindexes" : 2,
   "nchunks" : 26,
   "shards" : {
      "shard0000" : {
```

```
            "ns" : "kgc.sales",
            "count" : 60583,
            "size" : 6785296,
            "avgObjSize" : 112,
            "numExtents" : 7,
            "storageSize" : 22507520,
            "lastExtentSize" : 11325440,
            "paddingFactor" : 1,
            "paddingFactorNote" : "paddingFactor is unused and unmaintained in 3.0. It remains hard coded
                to 1.0 for compatibility only.",
            "userFlags" : 1,
            "capped" : false,
            "nindexes" : 2,
            "totalIndexSize" : 4349632,
            "indexSizes" : {
               "_id_" : 2796192,
               "pid_1" : 1553440
            },
            "ok" : 1
         },
         "shard0001" : {
            "ns" : "kgc.sales",
            "count" : 60734,
            "size" : 6802208,
            "avgObjSize" : 112,
            "numExtents" : 6,
            "storageSize" : 11182080,
            "lastExtentSize" : 8388608,
            "paddingFactor" : 1,
            "paddingFactorNote" : "paddingFactor is unused and unmaintained in 3.0. It remains hard coded
                to 1.0 for compatibility only.",
            "userFlags" : 1,
            "capped" : false,
            "nindexes" : 2,
            "totalIndexSize" : 5044592,
            "indexSizes" : {
               "_id_" : 2493680,
               "pid_1" : 2550912
            },
            "ok" : 1
         }
      },
      "ok" : 1
   }
   mongos>
```

分片几乎是平均分配，选择分片依据的列时，尽量选择不重复的列，不要选择例

如“性别”这种只有两个值的列。

2. 给分片添加标签

可以使用 sh.addShardTag() 命令给分片添加一个标签。

```
mongos> sh.addShardTag("shard0000","sales00")
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongos> sh.addShardTag("shard0001","sales01")
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
mongos> sh.status()
--- Sharding Status ---
......
  shards:
    {  "_id" : "shard0000",  "host" : "192.168.6.110:47017",  "tags" : [ "sales00" ] }
    {  "_id" : "shard0001",  "host" : "192.168.6.110:47018",  "tags" : [ "sales01" ] }
......
mongos>
mongos> sh.help()
```

可以使用 sh.help() 查看分片相关的命令。

3. 连接配置服务器

使用 mongo 命令连接到配置服务器。

```
mongos> exit
bye
[root@cent6s1 bin]# mongo --port 37017
MongoDB shell version: 3.2.10
connecting to: 127.0.0.1:37017/test
configsvr> show dbs
config  0.031GB
local   0.078GB
configsvr> use config
switched to db config
configsvr> show collections
actionlog
changelog
chunks
collections
databases
lockpings
locks
mongos
settings
shards
system.indexes
tags
```

```
version
```

配置服务器存储了 MongoDB 数据库集合分片的详细信息，可以通过以下命令查看。

```
configsvr> db.chunks.findOne()
{
  "_id" : "kgc.sales-pid_MinKey",
  "lastmod" : Timestamp(2, 0),
  "lastmodEpoch" : ObjectId("58383490b89ebe8bbfe76c5a"),
  "ns" : "kgc.sales",
  "min" : {
    "pid" : { "$minKey" : 1 }
  },
  "max" : {
    "pid" : 708
  },
  "shard" : "shard0001"
}
configsvr> db.collections.find()
{ "_id" : "kgc.sales", "lastmodEpoch" : ObjectId("58383490b89ebe8bbfe76c5a"), "lastmod" :
    ISODate("1970-02-19T17:02:47.321Z"), "dropped" : false, "key" : { "pid" : 1 }, "unique" : false }
{ "_id" : "kgc.sales2", "lastmodEpoch" : ObjectId("583847a8b89ebe8bbfeb2226"), "lastmod" :
    ISODate("1970-02-19T17:02:47.303Z"), "dropped" : false, "key" : { "qty" : 1 }, "unique" : false }
{ "_id" : "kgc.sales3", "lastmodEpoch" : ObjectId("5838490ab89ebe8bbfeb225e"), "lastmod" :
    ISODate("1970-02-19T17:02:47.321Z"), "dropped" : false, "key" : { "price" : 1 }, "unique" : false }
configsvr> db.databases.find()
{ "_id" : "kgc", "primary" : "shard0000", "partitioned" : true }
configsvr> db.mongos.find()
{ "_id" : "cent6s1:27017", "ping" : ISODate("2016-11-25T15:02:21.578Z"), "up" : NumberLong(12136),
    "waiting" : true, "mongoVersion" : "3.2.10" }
configsvr> db.settings.find()
{ "_id" : "chunksize", "value" : NumberLong(1) }
configsvr> db.shards.find()
{ "_id" : "shard0000", "host" : "192.168.6.110:47017", "tags" : [ "sales00" ] }
{ "_id" : "shard0001", "host" : "192.168.6.110:47018", "tags" : [ "sales01" ] }
configsvr>
```

4. 添加或删除 sharding server

根据需求可以添加或删除 sharding server。下面再添加一个 sharding server。

```
configsvr> exit
bye
[root@cent6s1 bin]# cp mongodb1.conf mongodb4.conf
[root@cent6s1 bin]# vi mongodb4.conf
port=47019
dbpath=/data/mongodb4
logpath=/data/logs/mongodb/mongodb4.log
```

```
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
shardsvr=true
[root@cent6s1 bin]# mkdir -p /data/mongodb4
[root@cent6s1 bin]# touch /data/logs/mongodb/mongodb4.log
[root@cent6s1 bin]# chmod 777 /data/logs/mongodb/mongodb4.log
[root@cent6s1 bin]# ./mongodb mongodb4 start
about to fork child process, waiting until server is ready for connections.
forked process: 31155
child process started successfully, parent exiting
[root@cent6s1 bin]#
```

使用 sh.addShard("192.168.6.110:47019") 命令添加一个新的分片服务器。再次查看分片信息，发现分片均匀地分布到 3 个分片服务器上。

```
[root@cent6s1 bin]# mongo
MongoDB shell version: 3.2.10
connecting to: test
mongos> sh.addShard("192.168.6.110:47019")
{ "shardAdded" : "shard0002", "ok" : 1 }
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("58382315b89ebe8bbfe3b6c7")
  }
  shards:
    { "_id" : "shard0000", "host" : "192.168.6.110:47017", "tags" : [ "sales00" ] }
    { "_id" : "shard0001", "host" : "192.168.6.110:47018", "tags" : [ "sales01" ] }
    { "_id" : "shard0002", "host" : "192.168.6.110:47019" }
  active mongoses:
    "3.2.10" : 1
  balancer:
    Currently enabled:  yes
    Currently running:  no
    Failed balancer rounds in last 5 attempts:  0
    Migration Results for the last 24 hours:
        48 : Success
        1 : Failed with error 'chunk too big to move', from shard0000 to shard0002
        5 : Failed with error 'chunk too big to move', from shard0000 to shard0001
  databases:
    { "_id" : "kgc", "primary" : "shard0000", "partitioned" : true }
        kgc.sales
```

```
        shard key: { "pid" : 1 }
        unique: false
        balancing: true
        chunks:
          shard0000   9
          shard0001   9
          shard0002   8
        too many chunks to print, use verbose if you want to force print
      kgc.sales2
        shard key: { "qty" : 1 }
        unique: false
        balancing: true
        chunks:
          shard0000   3
          shard0001   3
          shard0002   2
        { "qty" : { "$minKey" : 1 } } -->> { "qty" : 2 } on : shard0000 Timestamp(6, 1)
        { "qty" : 2 } -->> { "qty" : 3 } on : shard0000 Timestamp(1, 1) jumbo
        { "qty" : 3 } -->> { "qty" : 4 } on : shard0002 Timestamp(7, 0)
        { "qty" : 4 } -->> { "qty" : 5 } on : shard0001 Timestamp(7, 1)
        { "qty" : 5 } -->> { "qty" : 6 } on : shard0001 Timestamp(4, 0)
        { "qty" : 6 } -->> { "qty" : 7 } on : shard0001 Timestamp(5, 0)
        { "qty" : 7 } -->> { "qty" : 16 } on : shard0002 Timestamp(6, 0)
        { "qty" : 16 } -->> { "qty" : { "$maxKey" : 1 } } on : shard0000 Timestamp(1, 7)
      kgc.sales3
        shard key: { "price" : 1 }
        unique: false
        balancing: true
        chunks:
          shard0000   9
          shard0001   9
          shard0002   8
        too many chunks to print, use verbose if you want to force print

mongos>
```

使用 db.runCommand({"removeshard":"192.168.6.110:47019"}) 命令可以删除新添加的分片服务器。

```
mongos> use admin
switched to db admin
mongos> db.runCommand({"removeshard":"192.168.6.110:47019"})
{
  "msg" : "draining started successfully",
  "state" : "started",
  "shard" : "shard0002",
  "note" : "you need to drop or movePrimary these databases",
```

```
    "dbsToMove" : [ ],
    "ok" : 1
}
mongos> sh.status()
```

本章总结

- MongoDB 分片通过在多台服务器上分割数据，使得数据库系统能存储和处理更多的数据。
- MongoDB 分片群集主要有如下三个主要组件：Shard 分片服务器、Config Server 配置服务器，Routers 前端路由。
- 部署分片群集过程包括部署分片服务器，部署配置服务器、路由服务器，登录路由服务器，配置数据库集合分片等步骤。
- 片键的选择原则上要结合实际的业务需求，尽可能选择存储分片均匀的键。

本章作业

1. 部署一个简单的 MongoDB 分片群集环境。
2. 导入 sales 数据到 kgc 数据库的 sales 集合，对该集合进行分片，查看分片信息。
3. 添加分片服务器，再次查看分片信息。
4. 用课工场 APP 扫一扫，完成在线测试，快来挑战吧！



随手笔记



课工场岗位课程

- 大数据开发工程师
- 前端开发工程师
- Android 开发工程师
- PHP 开发工程师
- 新媒体运营师
- 互联网营销师
- 电子商务师
- 移动端 UI 设计师
- 网页 UI 设计师
- 互联网 UI 设计师
- 动漫设计师
- Python 开发工程师
- 云计算工程师
- VR 游戏设计师
- VR 游戏开发工程师
- VR 商用开发工程师
- 人工智能工程师

更多课程请访问 kgc.cn

联系我们

北京课工场教育科技有限公司

网址：kgc.cn

Q Q：800161516

邮箱：ke@kgc.cn

电话：010-88550007

地址：北京市海淀区成府路 207 号 B 座一层

课工场
kgc.cn

本 书 特 色

领先的实用技能——打造云计算工程师

真实的项目驱动——行业知识与专业设计相呼应

科学的训练模式——授人以渔而不只是作品临摹

便捷的学习体验——传统教材与二维码革命性携手

云计算工程师系列

- 云计算部署实战
- 云计算与网络安全
- Web 开发实战
- Python 开发向导
- 大型网站架构与自动化运维
- Docker 容器与虚拟化技术
- 数据库应用
- Web 服务器群集
- Linux 网络服务与 Shell 脚本攻略
- Linux 系统管理
- 网络原理与应用

微信号：Waterpub-Pro
唯一官方微信服务平台

销售分类：数据库技术 / 数据库应用

ISBN 978-7-5170-5378-1

定价：39.00 元